Spectrum™ Technology Platform

Version 2018.2.0

Dataflow Designer's Guide

Table of Contents

1 - Getting Started

In this section

Installing the Client Tools

The Spectrum™ Technology Platform client tools are applications that you use to administer your server and design and run dataflows and process flows. You must install your Spectrum™ Technology Platform server before installing the client tools.

Before installing, be sure to read the release notes. The release notes contains a list of known issues, important compatibility information, and release-specific installation notes.

This procedure describes how to install the following client tools:

• **Enterprise Designer**— Use Enterprise Designer to create, modify, and run dataflows.
• **Job Executor**—Job Executor is a command line tool that allows you to run a job from a command line or script. The job must have been previously created and saved on Spectrum™ Technology Platform using Enterprise Designer.
• **Process Flow Executor**—Process Flow Executor is a command line tool that allows the execution of a process flow from a command line or script. The process flow must have been previously created and saved on Spectrum™ Technology Platform using Enterprise Designer.
• **Administration Utility**—The Administration Utility provides command line access to several administrative functions. You can use it in a script, allowing you to automate certain administrative tasks. You can also use it interactively.

> **Note:** As of Spectrum version 11.0, Management Console is a web-based tool rather than installable client as it was in previous releases.

To install the client tools:

1. Open a web browser and go to the Spectrum™ Technology Platform Welcome Page at:

    http://<servername>:<port>

    For example, if you installed Spectrum™ Technology Platform on a computer named "myspectrumplatform" and it is using the default HTTP port 8080, you would go to:

    http://myspectrumplatform:8080

2. Click **Platform Client Tools**.
3. Download the client tool you want to install.

Starting Enterprise Designer

Enterprise Designer is a Windows application for creating dataflows. To start Enterprise Designer:

1. Select **Start** > **Programs** > **Pitney Bowes** > **Spectrum™ Technology Platform** > **Client Tools** > **Enterprise Designer**.

2. Type in the server name or IP address,or select it from the drop-down list. If you are using Spectrum™ Technology Platform in a cluster, enter the name of IP address of the cluster's load balancer.

3. Enter your user name and password.

4. In the Port field, enter the network port that the server has been configured to use for Spectrum™ Technology Platform communication. The default port number is 8080.

5. Click **Use secure connection** if you want communication between the client and the server to take place over an HTTPS connection.

   > **Note:** A secure connection is only available if HTTPS communication has been configured on the server. If you are running Enterprise Designer on Windows 7, using the IP address in the **Server name** field may not work, depending on the type of certificate used to secure the communication between Enterprise Designer and the server. If the IP address does not work, use the host name instead.

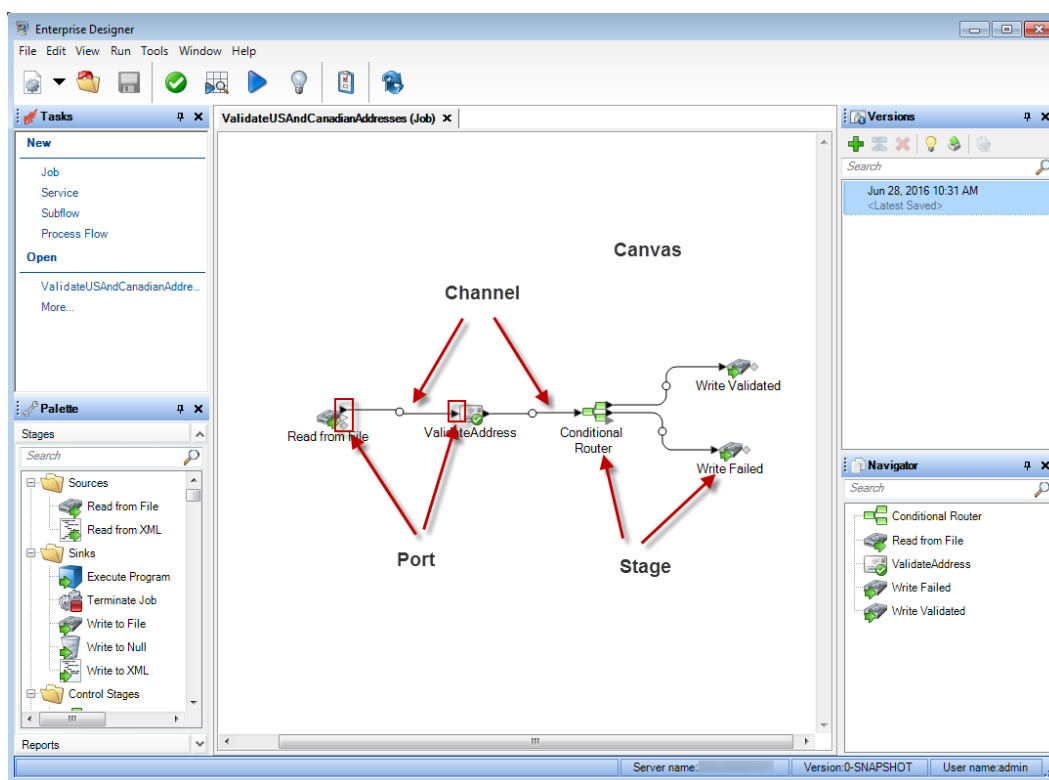6. Click **Login**.

A First Look at Enterprise Designer

Enterprise Designer is a visual tool for creating dataflows. Using this client, you can:

- Create and modify jobs, services, subflows, and process flows
- Test dataflows for problems
- Expose and hide services
- Generate reports

The Enterprise Designer window looks like this:

In order to work with dataflows you will need to understand a few important terms:

Canvas The canvas is the main work area. The picture above shows the canvas open with a dataflow named ValidateUSAndCanadianAddresses. It is a job dataflow, which means it performs batch processing by reading data from a file and writing output to a file. In this case, the dataflow is writing output to two files.

Stage Stages, represented by icons on the canvas, perform a specific type of activity, such as sorting records, validating addresses, matching similar records, and so on. To add a stage, drag the stage from the Palette (on the left side of the window) onto the canvas.

If a stage requires your attention, a blue circle appears on the icon:

A dataflow cannot run successfully if it has stages that require attention. So, double-click the stage to configure the required settings. Once you have configured all the required settings, the blue circle no longer appears:

Channel A channel is a connection between two or more stages through which records are passed from one stage to another. In the above example, you can see that the Read from File stage is connected to the ValidateAddress stage with a channel. Records are read into

the dataflow in Read from File then sent to ValidateAddress through this channel. ValidateAddress is then connected to Conditional Router through a channel. Conditional Router, which analyzes records and sends them along different paths in a dataflow depending on the conditions defined by the dataflow designer, has two channels going out of it, one to a Write Validated stage and one to a Write Failed stage.

The dot in the middle of a channel may change colors to indicate different conditions:

Red   Indicates an error, such as a type conversion failure that makes a field unusable by the downstream stage.
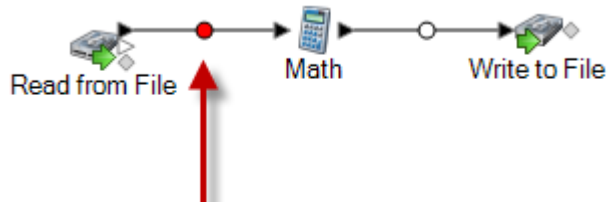
Yellow   You have removed a field that is needed by a downstream stage.
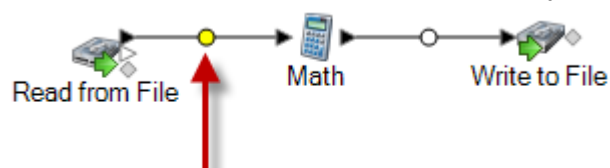
Blue   Automatic type conversion has successfully converted a field to the data type required by the downstream stage.
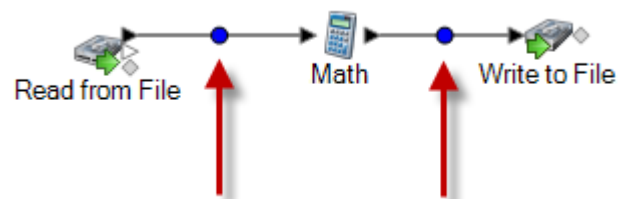
Black   A field is being renamed in the channel.
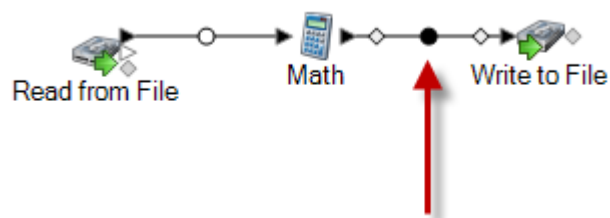
White No action is being taken on fields.
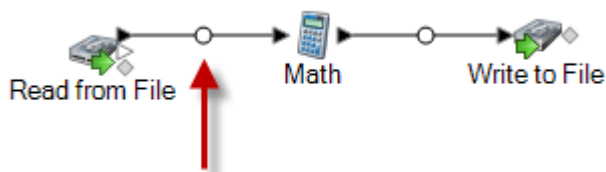
Port If you look closely at the stage icons you will notice small triangular or diamond shaped ports on the sides of each stage. A port is the mechanism by which a stage sends data into, or reads data from, a channel. Stages that read data into the dataflow (called "sources") only have output ports since they are always at the start of a dataflow. Stages that send data out of the dataflow (called "sinks") only have input ports since they are always at the end of a dataflow. All other stages have both input and output ports. In addition, some stages have error ports, which are used to output records that cause errors during the stage's processing, and some stages have report ports, which are used to generate reports about the stage's output.

In addition, the Enterprise Designer window has these features:

Feature	Description
Tasks	Provides a quick way to create a new job, service, subflow, or process flow. Also allows you to open dataflows that were recently open.
Server Explorer	Shows all the flows saved on the Spectrum™ Technology Platform server. If the server explorer this is not visible, select **View** > **Server Explorer**. You can organize flows into folders. To create a folder, right-click the server name and select **New Folder**. Flow names must be unique across all folders. You cannot have two flows with the same name even if they are in different folders.
Palette	Contains all the stages and reports you can add to your dataflow. The stages available in the palette depend on the modules you have licensed.
Canvas	The work area onto which you drag stages and connect them with channels to make dataflows. You can have several dataflow canvases open at once.
Versions	The Versions feature in Enterprise Designer allows you to keep a revision history of your dataflows. You can view previous versions of a dataflow, expose older versions for execution, and keep a history of your changes in case you ever need to revert to a previous version of a dataflow.

Feature	Description
Navigator	Lists the stages and reports in the flow. You can right-click an item in the **Navigator** pane to edit its options.

My First Dataflow (Job)

In this topic you will create a simple dataflow that reads data from a file, sorts it, then writes it to a file. Since this dataflow reads data from a file and writes its output to a file, it is a "job", which is a dataflow that performs batch processing. (The other primary type of dataflow, a "service", performs interactive processing via an API or web service call to the server.)

1. The first step will be to create some sample data to use as input to your dataflow. Using a text editor, create a file that looks like this:

```
FirstName,LastName,Region,Amount
Alan,Smith,East,18.23
Jeannie,Wagner,North,45.43
Joe,Simmons,East,10.87
Pam,Hiznay,Central,98.78
```

2. Save the file in a convenient location.
3. Select **Start** > **Programs** > **Pitney Bowes** > **Spectrum™ Technology Platform** > **Client Tools** > **Enterprise Designer**.
4. Select **File** > **New** > **Dataflow** > **Job**.
5. You are now ready to begin creating your dataflow. The first step is to define the input to the dataflow. To do this:

    a) Drag a Read from File stage onto the canvas:
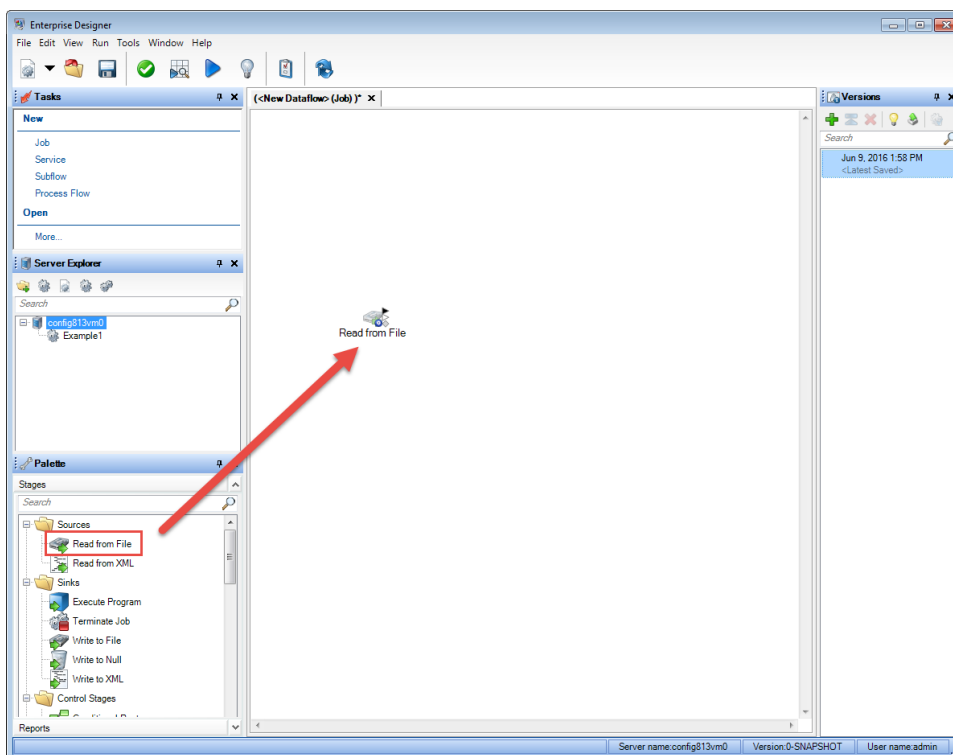
b) Double-click the Read from File stage on the canvas.

c) In the **File name** field, specify the file you created in step **1** on page 10.

d) In the **Record type** field, choose **Delimited**.

e) In the **Field separator** field, select **Comma (,)**.

f) Check the **First row is header record** box.

g) Click the **Fields** tab.

h) Click **Regenerate** then click **Yes**.

   The stage is automatically configured for the fields in your input file.

i) Click **Detect Type**. This scans the input file and determines the appropriate data type for each field. Notice that the type for the **Amount** filed changes from string to double.

j) You have finished configuring Read from File. Click **OK**.

6. Next, you will add a stage that will sort the records by region. To do this:

a) Drag the Sorter stage onto the canvas

b) Click the solid black triangle on the right side of the Read from File stage (the output port) and drag it to the left side of the Sorter stage on the canvas to create a channel connecting Read from File and Sorter.

   Your dataflow should look like this:
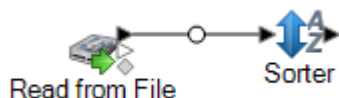
c) Double-click the Sorter stage on the canvas.

d) Click **Add**.

e) In the **Field Name** field, select **Region**.

f) You have finished configuring Sorter. Click **OK**.

7. Finally, you will define the output file where the dataflow will write its output. To do this:

a) Drag a Write to File stage onto the canvas.

b) Click the solid black triangle on the right side of the Sorter stage and drag it to the left side of the Write to File stage on the canvas.

   Your dataflow should look like this:

c) Double-click the Write to File stage.

d) In the **File name** field, specify an output file. This can be any file you want.

e) In the **Field separator** field, select **Comma (,)**.

f) Check the **First row is header record** box.

g) Click the **Fields** tab.

h) Click **Quick Add**.

i) Click **Select All** then click **OK**.

j) Using the **Move Up** and **Move Down** buttons, reorder the fields so that they are in the following order:

   FirstName
   LastName
   Region
   Amount

   This will make the records in your output file have the fields in the same order as your input file.

k) You have finished configuring Write to File. Click **OK**.

8. In Enterprise Designer, select **File** > **Save**.

9. Give your dataflow a name and click **OK**.

10. Your dataflow is now ready to run. Select **Run** > **Run Current Flow**.

11. The **Execution Details** window appears and shows the status of the job. Click **Refresh**. Once the status shows **Succeeded** click **Close**.

Open the output file you specified in the Write to File stage. You will see that the records have been sorted by region as you specified in the Sorter stage.

```
FirstName,LastName,Region,Amount
Pam,Hiznay,Central,98.78
```

```
Alan,Smith,East,18.23
Joe,Simmons,East,10.87
Jeannie,Wagner,North,45.43
```

Congratulations! You have designed and executed your first job dataflow.

My First Dataflow (Service)

In this topic you will create a simple dataflow that accepts data from an API or web service call, processes the data, and returns a response via the API or web service. Since this dataflow is intended to be exposed as a service on the Spectrum™ Technology Platform server, it is a "service" dataflow. (The other primary type of dataflow, a "job", performs batch processing, reading data from a file or database, processing the data, then writing the output to a file or database.)

1. Select **Start** > **Programs** > **Pitney Bowes** > **Spectrum™ Technology Platform** > **Client Tools** > **Enterprise Designer**.
2. Select **File** > **New** > **Dataflow** > **Service**.
3. You are now ready to begin creating your dataflow. The first step is to define the input to the dataflow. Your datafow will take two fields as input: FirstName and LastName.

   a) Drag an Input stage from the palette to the canvas.
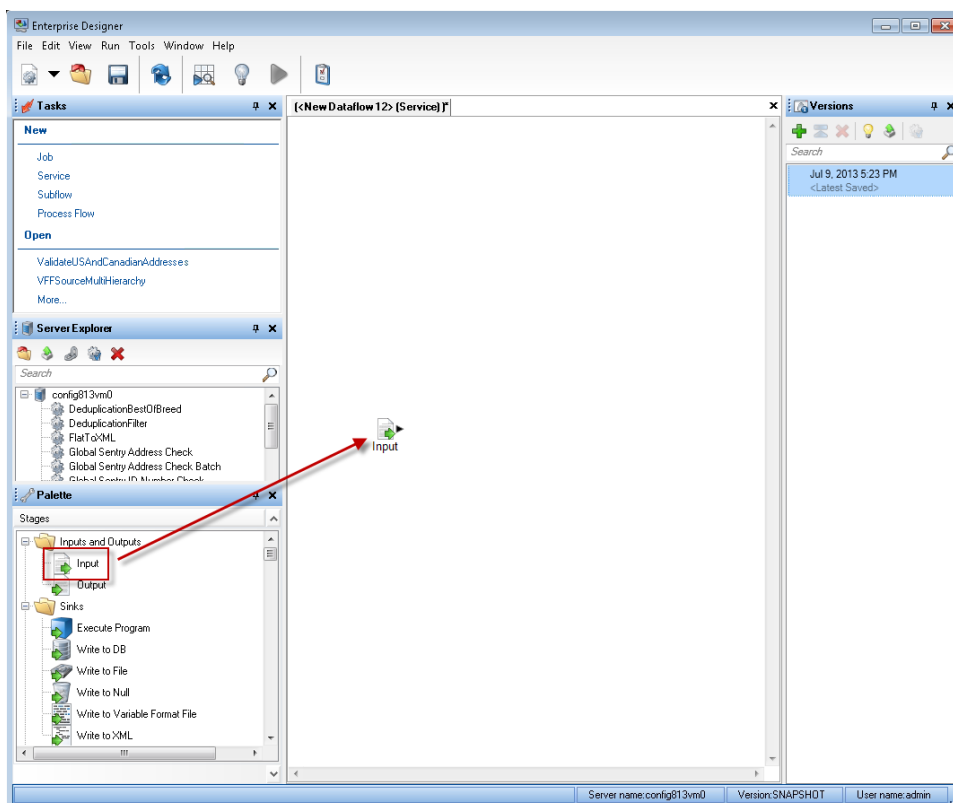
b) Double-click the Input stage on the canvas.

c) Click **Add** then click **Add** again.

d) In the **Field name** field, type `FirstName`.

e) Click **OK**, then click **OK** again.

f) Click **Add** then click **Add** again.

g) In the **Field name** field, type `LastName`.

h) Click **OK**, then click **OK** again.

i) You have finished defining the dataflow input. Click **OK**.

4. Next, you will add a stage to change the casing of the data in the FirstName and LastName fields to all upper case.

a) Drag a Transformer stage from the palette to the canvas.

b) Click the solid black triangle on the right side of the Input stage (the output port) and drag it to the left side of the Transformer stage on the canvas to create a channel connecting Input and Transformer.

Your dataflow should look like this:
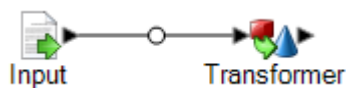
c) Double-click the Transformer stage.

d) Click **Add**.

e) In the tree on the left side, under **Formatting** click **Case**.

f) In the **Field** field, select **FirstName**. Leave **Upper** selected.

g) Click **Add**.

h) In the **Field** field, select **LastName**. Leave **Upper** selected.

i) Click **Add**.

j) Click **Close**.

k) You have finished configuring Transformer to change the value in the FirstName and LastName fields to upper case. Click **OK**.

5. Finally, you will define the output for the dataflow. Your dataflow will return the FirstName and LastName fields as output.

a) Drag an Output stage onto the canvas.

b) Click the solid black triangle on the right side of the Transformer stage and drag it to the left side of the Output stage on the canvas.

Your dataflow should look like this:
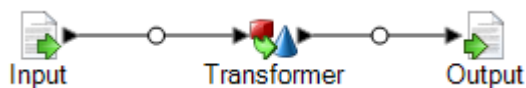
c) Double-click the Output stage on the canvas.

d) Check the **Expose** box. The check boxes next to FirstName and LastName should now be checked.

e) Click **OK**.

6. In Enterprise Designer, select **File** > **Save**.

7. Give your dataflow the name `MyFirstDataflow-Service` and click **OK**.

8. Select **File** > **Expose/Unexpose and Save**. This exposes your dataflow, making it available as a service on the server.

9. To test your service:

a) Open Management Console by going to this URL in a web browser:

http://*server*:*port*/managementconsole

Where *server* is the server name or IP address of your Spectrum™ Technology Platform server and *port* is the HTTP port used bySpectrum™ Technology Platform. By default, the HTTP port is 8080.

b) Go to **Services** > **Other Services**.

c) In the list of services, check the box next to **MyFirstDataflow-Service** then click the Edit button ✏.

d) Enter a name in the FirstName field in all lower case letters.

e) Enter a name in the LastName field in all lower case letters.

f) Click **Run Preview**.

You can see that the service made the name fields all upper case letters, as you specified in your dataflow's Transformer stage.
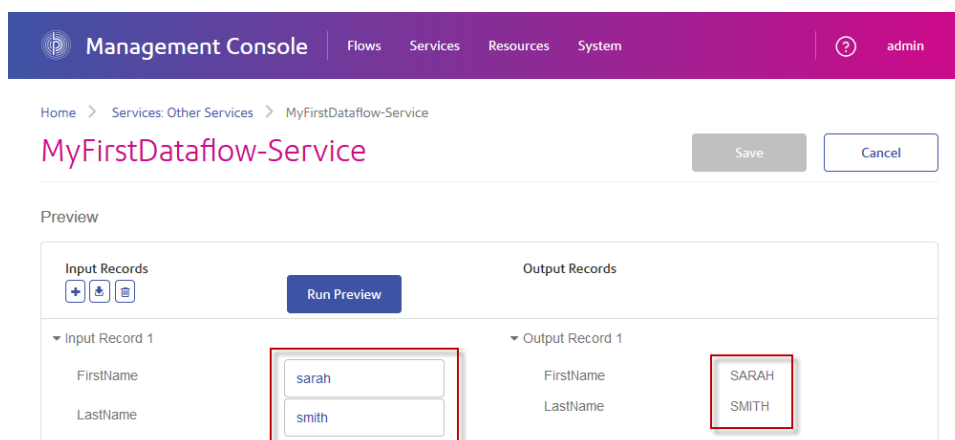
Congratulations! You have designed and executed your first service dataflow. The service is now available on the server and can be accessed via an API or web services call. The resource URL for this service's SOAP endpoint is:

```
http://<ServerName>:<Port>/soap/MyFirstDataflow-Service
```

The resource URL for this service's REST endpoint is:

```
http://<ServerName>:<Port>/rest/MyFirstDataflow-Service
```

Dataflow Templates

Dataflow templates illustrate ways in which you can use Spectrum™ Technology Platform and its modules to meet your business needs. They show how particular modules solve various requirements, such as parsing, standardizing, and validating names and addresses, geocoding addresses, and so on.

Dataflow templates are delivered with each module that you license. For instance, if you are licensed for the Data Normalization Module, you receive the Standardizing Personal Names dataflow template. If you are licensed for the Universal Addressing Module, you receive the Validating U.S. and Canadian Addresses dataflow templates.

Depending on the purpose of each template, it may be a job with sample data or it may be a service with no sample data. You can use dataflows in their original state and run those that are delivered as jobs to see how they function. Alternatively, you can manipulate the dataflows by changing input and output files or by bringing services into your own jobs and adding input and output files.

> **Note:** These samples are intended as illustrations of various Spectrum™ Technology Platform features. They are not intended to be complete solutions to your particular business environment.

Creating a Dataflow Using a Template

Dataflow templates are delivered with each module that you license. To create a dataflow using a template,

- In Enterprise Designer go to **File** > **New** > **Dataflow** > **From Template**.
- Or, you can click the **New** icon and select New Dataflow From Template.

A list of templates available for the modules you have installed is displayed.

Importing and Exporting Dataflows

You can exchange dataflows with other Enterprise Designer users with the import and export features.

> **Note:** Dataflows can only be exchanged between identical versions of Spectrum™ Technology Platform.

- To export a dataflow, select **File** > **Export**. If you have used the Versions feature to save versions of the dataflow, the version you have currently selected is the version that is exported.

   > **Note:** Do not use special characters in the name of the services and jobs you define. Doing so may result in an error during export.

- To import a process flow, select **File** > **Import** > **Process Flow**.
- To import a dataflow, select **File** > **Import** > **Dataflow**. The stages in the dataflow must be available on your system before you import the dataflow. If the dataflow you import contains unavailable stages, you will see an error.
- If you use Server Explorer to organize your dataflows you can also export a dataflow by right-clicking it and selecting **Export**. To import a dataflow using Server Explorer, right-click in the location in Server Explorer where you want to import the dataflow and select **Import**.

2 - Designing a Flow

In this section

Types of Flows

A dataflow is a series of operations that takes data from some source, processes that data, then writes the output to some destination. The processing of the data can be anything from simple sorting to more complex data quality and enrichment actions. The concept of a dataflow is simple, but you can design very complex dataflows with branching paths, multiple sources of input, and multiple output destinations.

There are four types of dataflows: jobs, services, subflows, and process flows.

Job

A job is a dataflow that performs batch processing. A job reads data from one or more files or databases, processes that data, and writes the output to one or more files or databases. Jobs can be executed manually in Enterprise Designer or can be run from a command line using the job executor.

The following dataflow is a job. Note that it uses the Read from File stage for input and two Write to File stages as output.
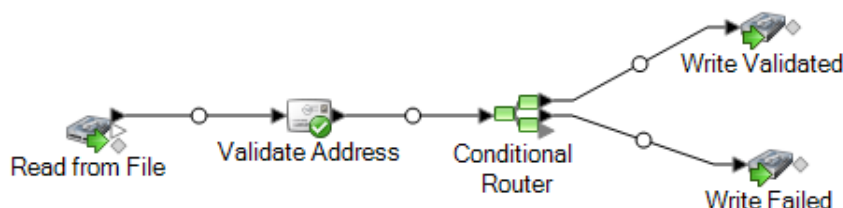
Service

A service is a dataflow that you can access as web services or using the Spectrum™ Technology Platform API. You pass a record to the service and optionally specify the options to use when processing the record. The service processes the data and returns the data.

Some services become available when you install a module. For example, when you install the Universal Addressing Module the service ValidateAddress becomes available on your system. In other cases, you must create a service in Enterprise Designer then expose that service on your system as a user-defined service. For example, the Location Intelligence Module's stages are not available as services unless you first create a service using the module's stages.

You can also design your own custom services in Enterprise Designer. For example, the following dataflow determines if an address is at risk for flooding:
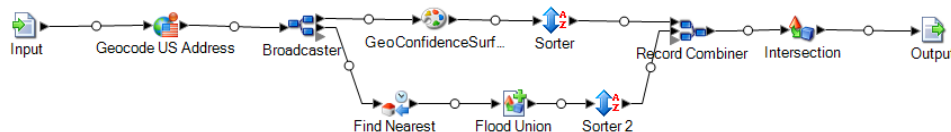
Note: Since the service name, option name, and field name ultimately become XML elements, they may not contain characters that are invalid in XML element names (for example, spaces are not valid). Services not meeting this requirement will still function but will not be exposed as web services.

Subflow

A subflow is a dataflow that can be reused within other dataflows. Subflows are useful when you want to create a reusable process that can be easily incorporated into dataflows. For example, you might want to create a subflow that performs deduplication using certain settings in each stage so that you can use the same deduplication process in multiple dataflows. To do this you could create a subflow like this:

You could then use this subflow in a dataflow. For example, you could use the deduplication subflow within a dataflow that performs geocoding so that the data is deduplicated before the geocoding operation:

In this example, data would be read in from a database then passed to the deduplication subflow, where it would be processed through Match Key Generator, then Intraflow Match, then Best of Breed, and finally sent out of the subflow and on to the next stage in the parent dataflow, in this case Geocode US Address. Subflows are represented as a puzzle piece icon in the dataflow, as shown above.

Subflows that are saved and exposed are displayed in the **User Defined Stages** folder in Enterprise Designer.

Process Flow

A process flow executes a series of activities such as jobs and external applications. Each activity in the process flow executes after the previous activity finishes. Process flows are useful if you want to run multiple dataflows in sequence or if you want to run an external program. For example, a process flow could run a job to standardize names, validate addresses, then invoke an external application to sort the records into the proper sequence to claim postal discounts. Such a process flow would look like this:

In this example, the jobs Standardize Names and Validate Addresses are exposed jobs on the Spectrum™ Technology Platform server. Run Program invokes an external application, and the Success activity indicates the end of the process flow.

Flow Input

To define the input for a dataflow, use a "source" stage. A source is the first stage in a dataflow. It defines the input data you want to process.

Input for a Job

Input data for a job can come from a file or a database. Spectrum™ Technology Platform has the ability to read data from many file formats and database types. The types of data sources you can read from depend on which modules you have licensed. The Enterprise Data Integration Module provides access to the most data sources of any module.

> **Note:** When designing a job, it is a good idea to account for the possibility of malformed input records. A malformed record is one that cannot be parsed using one of the parser classes provided by Spectrum™ Technology Platform. For information about handling malformed input records, see **Managing Malformed Input Records** on page 21.

Input for a Service

Input data for a service is defined in an Input stage. This stage defines the fields that the service will accept from a web service request or an API call.

Defining Job Input

Input data for a job can come from a file, database, or cloud service, depending on the modules you have licensed. Each module supports input from different sources, and the procedure for configuring each type of source varies greatly. See the solution guide for your modules available at **support.pb.com/spectrum**.

Managing Malformed Input Records

A malformed record is one that Spectrum™ Technology Platform cannot parse. When Spectrum™ Technology Platform encounters a malformed record, it can do one or more of the following:

- Terminate the job
- Continue processing
- Continue processing until a certain number of bad records are encountered

- Continue processing but write bad records to a log file (via an optional sink stage)

    Note: Malformed records functionality is limited to sources configured to read from files local to the server and that do not have sorting configured. When a source is configured with either a remote file or with sort fields and the source encounters a malformed record, the job will terminate regardless of the configuration for malformed records.

To manage malformed records,

1. Open the job in Enterprise Designer.
2. Add a malformed records sink in your dataflow.
    a) Create your job by defining your input file and source stage and adding services and subflows to your dataflow.
    b) Do one of the following:

        - Connect a sink stage to the optional output port on the source stage in your dataflow. The optional port is the clear output port just beneath the black output port on your source stage. If you mouse over this port, you will see a tool tip that says, "error_port." Malformed records are written to this sink.

        - Connect nothing to the optional output port on the source stage in your dataflow, which causes Spectrum™ Technology Platform to ignore malformed records.

    The completed dataflow should look something like this:
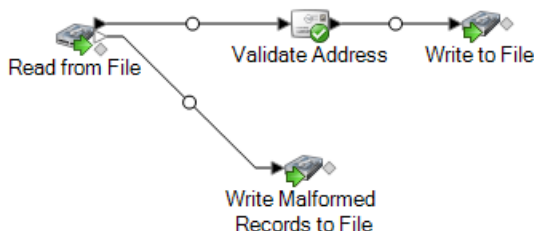
    When you run your job, the Execution History will contain a column that shows the number of malformed records that were encountered during the job.

3. By default Spectrum™ Technology Platform will terminate a job when it encounters a malformed record. This default behavior can be changed in Management Console. Regardless of your system's defualt behavior, you can override the default behavior for a job by following these steps:
    a) Open the job in Enterprise Designer.
    b) Within an open job, go to **Edit** > **Job Options**.
    c) Select either **Do not terminate the job on a malformed record** or select **Terminate the job after encountering this many malformed records** and enter the number of malformed records you will allow a job to encounter before terminating.

Defining Service Input

The Input stage defines the input fields for a service or subflow. It also defines test data to use during data inspection.

Input Fields Tab

This tab lists the fields that the dataflow accepts as input. If the Input stage is connected to another stage in the dataflow, a list of fields used by the stages in the dataflow is shown. For more information, see **Defining Input Fields for a Service or Subflow** on page 23.

Inspection Input Tab

This tab allows you to specify test input records to use with the Data Inspection tool. For more information about data inspection, see **Inspecting a Dataflow** on page 138.

Defining Input Fields for a Service or Subflow

To define the input fields for a service or subflow, use the Input stage.

> **Note:** If you define hierarchical data in the input fields, you will not be able to import data or view the data vertically.

1. Drag an Input stage to the canvas.
2. Connect the Input stage to the following stage in the dataflow.
3. Double-click the Input stage.
4. Select the fields you want to use for input. The list of fields shown depends on the stage that the Input stage is connected to.
5. To add a new field to the field list, click **Add**. The **Add Custom Field** window appears.
6. Click **Add** again.
7. In the **Field name** field, enter the name you want to use for this field.
8. Select the data type.

   The following data types are supported:

   bigdecimal   A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.

   boolean   A logical type with two values: true and false.

   bytearray   An array (list) of bytes.

   > **Note:** Bytearray is not supported as an input for a REST service.

date	A data type that contains a month, day, and year. For example, 2012-01-30 or January 30, 2012. You can specify a default date format in Management Console.
datetime	A data type that contains a month, day, year, and hours, minutes, and seconds. For example, 2012/01/30 6:15:00 PM.
double	A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52}) \times 2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.
float	A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23}) \times 2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.
integer	A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647).
list	Strictly speaking, a list is not a data type. However, when a field contains hierarchical data, it is treated as a "list" field. In Spectrum™ Technology Platform a list is a collection of data consisting of multiple values. For example, a field Names may contain a list of name values. This may be represented in an XML structure as:

```
<Names>
    <Name>John Smith</Name>
    <Name>Ann Fowler</Name>
</Names>
```

	It is important to note that the Spectrum™ Technology Platform list data type different from the XML schema list data type in that the XML list data type is a simple data type consisting of multiple values, whereas the Spectrum™ Technology Platform list data type is similar to an XML complex data type.
long	A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807).
string	A sequence of characters.
time	A data type that contains the time of day. For example, 21:15:59 or 9:15:59 PM.

You can also add a new, user-defined data type if necessary, and that new type can be a list of any defined data type. For example, you could define a list of names (string), or a new data type of addresses that includes AddressLine1 (string), City (string), StateProvince (string) and PostalCode (string). After you create the field, you can view the data type by accessing the Input Options dialog and pressing the button in the Data Type column. The **Data Type Details** dialog box will appear, showing the structure of the field.

9. Press **OK** again.

10. Click the **Expose** column check box to make the field available for stage operations. Clearing the check box and clicking **OK** deletes the field from the field list.

11. The **Data type name** field displays the default element name to use for input records in SOAP and REST web service requests to this service. The default value is `Row`. If you want to use a different element name for input records, enter it here.

    For example, with the default value `Row`, a JSON web service request would use `Row` as the element name for the input record, as shown here:

    ```
    {
      "Input":
      {
        "Row": [
          {
            "AddressLine1": "1825 Kramer Ln",
            "City": "Austin",
            "StateProvince": "TX"
          }
        ]

      }
    }
    ```

    If you were to change the value in the **Data type name** field to `Address`, the JSON request would need to use `Address` instead of `Row` as the element name for the record, as shown here:

    ```
    {
      "Input":
      {
        "Address": [
          {
            "AddressLine1": "1825 Kramer Ln",
            "City": "Austin",
            "StateProvince": "TX"
          }
        ]

      }
    }
    ```

Defining A Web Service Data Type

The **Data type name** field allows you to control the WSDL (SOAP) and WADL (REST) interfaces for the service you are creating. The name of the Rows element is determined by the name you give this stage in the service, and the name of the Row element is determined by the text you enter here.

> **Note:** For WSDL, both requests and responses are affected, but for WADL only responses are affected.

Prior to naming this stage and entering text in this field, your code might look like this:

```
<Rows>
    <Row>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
    </Row>
    <Row>
        <FirstName>Jane</FirstName>
    <LastName>Doe></LastName>
    </Row>
</Rows>
```

After naming this stage and entering text in this field, your code might look like this:

```
<Names>
    <Name>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
    </Name>
    <Name>
        <FirstName>Jane</FirstName>
        <LastName>Doe></LastName>
    </Name>
</Names>
```

Fields

Flat and Hierarchical Data

Spectrum™ Technology Platform supports flat data and hierarchical data. In general you can use either flat or hierarchical data as input and output for a dataflow. A few stages in the Enterprise Routing Module require data to be in a hierarchical format.

Flat Data

Flat data consists of records, one on each line, and fields in each record. Fields are delimited by a specific character or positioned in a defined location on the line. For example, this is flat data with comma-delimited fields:

```
Sam,43,United States
Jeff,32,Canada
Mary,61,Ireland
```

To read flat data into a dataflow, you can use the Read from File, Read from DB, or Input stages. To write flat data output from a dataflow, you can use the Write to File, Write to DB, or Output stages.

Hierarchical Data

Hierarchical data is a tree-like structure with data elements that have parent/child relationships. Spectrum™ Technology Platform can read and write hierarchical data in XML and Variable Format File format. For example, this shows hierarchical data in XML:

```
<customers>
 <customer>
  <name>Sam</name>
  <age>43</age>
  <country>United States</country>
 </customer>
 <customer>
  <name>Jeff</name>
  <age>32</age>
  <country>Canada</country>
 </customer>
 <customer>
  <name>Mary</name>
  <age>61</age>
  <country>Ireland</country>
 </customer>
</customers>
```

This example shows a structure where `<customer>` represents a record and each record consists of simple XML elements (`<name>`, `<age>`, and `<country>`).

Converting Data

There are many cases where you might need to convert data from flat to hierarchal, or from hierarchical to flat. For example, you may have data flow input in hierarchical format but want the data flow to output flat data. You may also need to convert flat input data to hierarchical data for certain stages (especially stages in the Location Intelligence Module) then convert the data back to flat data for output.

To convert data from flat to hierarchical you can use the following:

• The Process List tool
• The Aggregator stage in a dataflow

To convert data from hierarchical to flat use the Splitter stage.

Converting Flat Data to a List

Process List is a tool you can use within a service or subflow to turn flat data into a list. This is useful if your dataflows include stages that require list input, such as those in the Location Intelligence Module.

1. With an existing dataflow in place, right-click the stage whose output you want to convert into a list. This could be any stage except Input or Output.

2. Select **Process List**. You will see the stage within a blue square background.

3. To move a stage into and out of the process list, press the **Shift** key while dragging the additional stage.

   > **Note:** If you have several stages whose data you would like Process List to handle, consider creating a subflow, bringing it into your dataflow, and applying the Process List feature to the subflow as a whole.

4. The input and output fields of a process list are called "ListField." Using the Rename Fields function, you must map your input stage field to "ListField" in the input channel, and map "ListField" to your output stage field. For more information, see **Changing a Field's Name** on page 41.

5. If you want the list to keep the data in the same order in which it was input, right-click the Process List box and select Options. Then check the Maintain sort order box.

6. To confirm that the data input into the next stage will be formatted as a list, validate or inspect the dataflow. For more information on inspecting data, see **Inspecting a Dataflow** on page 138.

Data Types

Spectrum™ Technology Platform supports a variety of numeric, string, and complex data types. Depending on the type of processing you want to perform you may use one or more of these. For an address validation dataflow you might only use string data. For dataflows that involve the mathematical computations you may use numeric or Boolean data types. For dataflows that perform spatial processing you may use a complex data type. For dataflows that combine these, you may use a variety of data types.

Spectrum™ Technology Platform supports the following data types.

bigdecimal   A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.

boolean   A logical type with two values: true and false.

bytearray   An array (list) of bytes.

> **Note:** Bytearray is not supported as an input for a REST service.

date   A data type that contains a month, day, and year. For example, 2012-01-30 or January 30, 2012. You can specify a default date format in Management Console.

datetime   A data type that contains a month, day, year, and hours, minutes, and seconds. For example, 2012/01/30 6:15:00 PM.

double	A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52})\times2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.
float	A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23})\times2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.
integer	A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647).
list	Strictly speaking, a list is not a data type. However, when a field contains hierarchical data, it is treated as a "list" field. In Spectrum™ Technology Platform a list is a collection of data consisting of multiple values. For example, a field Names may contain a list of name values. This may be represented in an XML structure as:

```
<Names>
    <Name>John Smith</Name>
    <Name>Ann Fowler</Name>
</Names>
```

	It is important to note that the Spectrum™ Technology Platform list data type different from the XML schema list data type in that the XML list data type is a simple data type consisting of multiple values, whereas the Spectrum™ Technology Platform list data type is similar to an XML complex data type.
long	A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807).
string	A sequence of characters.
time	A data type that contains the time of day. For example, 21:15:59 or 9:15:59 PM.

Specifying a Field's Data Type

You can specify the data type for a field in these situations:

- **Source stages:** Specifying data types allows you to set the data type at the beginning of a dataflow, eliminating the need for data type conversions later in the dataflow. Note that for Read from DB, the data type is selected automatically and cannot be changed.
- **Sink stages:** Specifying data types allows you to control the data format returned by the dataflow. Note that for Write to DB, the data type is selected automatically and cannot be changed.
- **Transformer stage:** You can specify data types in this stage if you use a custom script.
- **Math stage and Group Statistics stage:** Since these stages perform mathematical calculations, choosing to use a particular numeric data type can have an effect on the results of the calculations, such as the precision of a division operation. If you specify a data type for a field that is different than the data type of the field coming into the stage, the downstream channel will automatically convert the field to the data type you specify, as described in **Automatic Data Type Conversion** on page 30.

Note: Each stage supports different data types. For a description of the supported data types for each stage, see the documentation for the stage.

Automatic Data Type Conversion

When the data presented to a stage is of an inappropriate type, Spectrum™ Technology Platform can, in some cases, automatically convert the data to the appropriate type. For example, Validate Address accepts only string data as input. If the PostalCode input field is of type integer, Spectrum™ Technology Platform can automatically convert the field to string and successfully process the PostalCode field. Likewise, the Math stage needs data to be of a numeric data type. If the incoming data is of type string, Spectrum™ Technology Platform can convert the data to the data type specified in the Math stage's Fields tab.

Automatic data type conversions happen in the channels of a dataflow. If a channel is successfully converting a data type, there will be a blue dot in the middle of the channel:
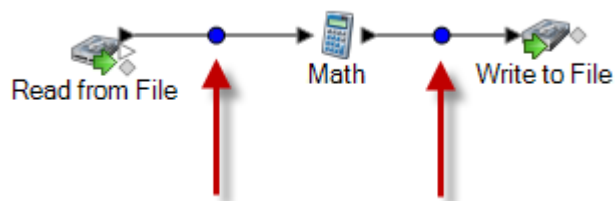
If you double-click the channel you can see the data type conversion that's occurring. In this case, string data is being converted to integer data:
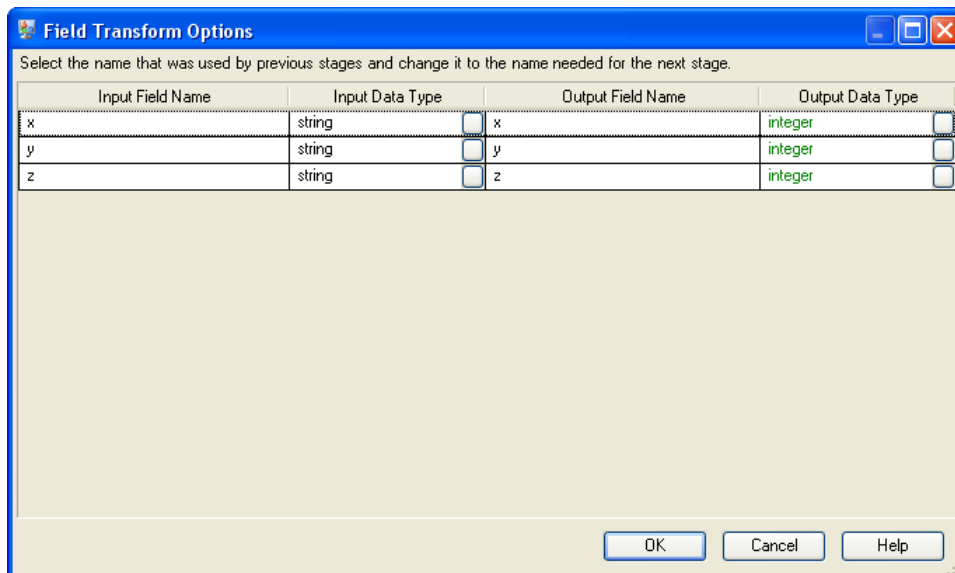
Note that you cannot change the data type in this dialog box for automatic data type conversions. The output data type is determined by settings in the downstream stage.

Fields that do not contain valid values or that cannot be converted result in a red circle in the channel.
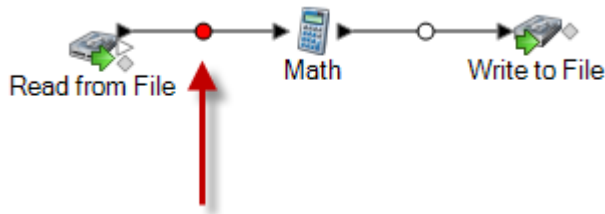
You can specify what the dataflow should do if type conversion fails by using the type conversion options.

Setting Data Type Conversion Options for a Dataflow

Data type conversion occurs when a dataflow automatically converts a field to the data type needed by a stage. Data type conversion also occurs when within some stages. For example, in Read from DB you can choose to have a field use the string data type even though the source data is in a numeric data type. The data is converted into the string data type when it is read into the dataflow.

There are two settings that you can use to control data type conversions. First, there are settings that determine how to format numeric, date, and time data converted into a string. For example, you may want date data that is converted into a string to be represented in the format mm/dd/yyyy rather than dd/mm/yyyy. The other setting controls what should happen if the system is unable to convert a field from one data type to another.

The default data type conversion settings for your system are specified in Management Console. You can override the default formats for individual dataflows in Enterprise Designer.

This procedure describes how to override the default data type conversion options for a dataflow.

> **Note:** Subflows inherit the type conversion settings from the dataflow they are in. You cannot specify type conversion settings for subflows.

1. Open the dataflow in Enterprise Designer.
2. Select **Edit** > **Type Conversion Options**.
3. Check the box **Override system default options with the following values**.
4. In the **Failure handling** field, specify what to do when a field's value cannot be automatically converted to the data type required by a stage.

Fail the dataflow	If a field cannot be converted the dataflow will fail.
Fail the record	If a field cannot be converted the record will fail but the dataflow will continue to run.
Initialize the field using default values	If a field cannot be converted the field's value is replaced with the value you specify here. This option is useful if you know that some records contain bad data and you want to replace the bad data with a default value. Specify a value for each data type.

5. Specify the formats that you want to use for date and time data that is converted to a string. When the data or time is converted to a string, the string will be in the format you specify here.

a) In the **Locale** field, select the country whose format you want to use for dates converted to a string. Your selection will determine the default values in the **Date**, **Time**, and **DateTime** fields. Your selection will also determine the language used when a month is spelled out. For example, if you specify English the first month of the year would be "January" but if you specify French it would be "Janvier."

b) In the **Date** field, select the format to use for date data when it is converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **M/D/YY** and a date field contains 2012-3-2, that date data would be converted to the string `3/2/12`.

c) In the **Time** field, select the format to use for time data when it is converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **h:mm a** and a time field contains 23:00, that time data would be converted to the string `11:00 PM`.

d) In the **DateTime** field, select the format to use for fields containing the DateTime data type when converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **M/d/yy h:mm a** and a DateTime field contains 2012-3-2 23:00, that DateTime data would be converted to the string `3/2/12 11:00 PM`.

e) In the **Whole numbers** field, select the formatting you want to use for whole numbers (data types float and double).

For example, if you choose the format **#,###** then the number 4324 would be formatted as `4,324`.

> **Note:** If you leave this field blank, numbers will be formatted in the same way they were in Spectrum™ Technology Platform 8.0 and earlier. Specifically, no thousands separator is used, the dot (".") is used as the decimal separator, numbers less than 10^{-3} or greater than or equal to 10^{7} are shown in scientific notation, and negative numbers have a minus sign ("-") in front of them. Also note that if you leave this field blank, numbers that use the bigdecimal data type will always be in the format #,###.000.

f) In the **Decimal numbers** field, select the formatting you want to use for numbers that contain a decimal value (data types integer and long).

For example, if you choose the format **#,##0.0#** then the number 4324.25 would be formatted as `4,324.25`.

> **Note:** If you leave this field blank, numbers will be formatted in the same way they were in Spectrum™ Technology Platform 8.0 and earlier. Specifically, no thousands separator is used, the dot (".") is used as the decimal separator, numbers less than 10^{-3} or greater than or equal to 10^{7} are shown in scientific notation, and negative numbers have a minus sign ("-") in front of them. Also note that if you leave this field

blank, numbers that use the bigdecimal data type will always be in the format #,###.000.

You can also specify your own date, time, and number formats if the ones available for selection do not meet your needs. To specify your own date or time format, type the format into the field using the notation described in **Date and Time Patterns** on page 33. To specify your own number format, type the format into the file using the notation described in **Number Patterns** on page 36.

6. Under **Null handling**, choose what to do if a field that needs type conversion contains a null value. If you select any of the following options, either the dataflow or the record containing the null value will fail based on whether you selected **Fail the dataflow** or **Fail the record** under **Type Conversion Failures**.

Fail null string	Fail the dataflow or record if type conversion is needed on a string field that contains a null value.
Fail null Boolean	Fail the dataflow or record if type conversion is needed on a Boolean field that contains a null value.
Fail null numeric	Fail the dataflow or record if type conversion is needed on a numeric field that contains a null value. Numeric fields include double, float, long, integer, and Big Decimal fields.
Fail null date	Fail the dataflow or record if type conversion is needed on a date field that contains a null value. This includes date, time, and DateTime fields.

Date and Time Patterns

When defining data type options for date and time data, you can create your own custom date or time pattern if the predefined ones do not meet your needs. To create a date or time pattern, use the notation described in the following table. For example, this pattern:

dd MMMM yyyy

Would produce a date like this:

14 December 2012

Letter	Description	Example
G	Era designator	AD
yy	Two-digit year	96
yyyy	Four-digit year	1996

Letter	Description	Example
M	Numeric month of the year.	7
MM	Numeric month of the year. If the number is less than 10 a zero is added to make it a two-digit number.	07
MMM	Short name of the month	Jul
MMMM	Long name of the month	July
w	Week of the year	27
ww	Two-digit week of the year. If the week is less than 10 an extra zero is added.	06
W	Week of the month	2
D	Day of the year	189
DDD	Three-digit day of the year. If the number contains less than three digits, zeros are added.	006
d	Day of the month	10
dd	Two-digit day of the month. Numbers less than 10 have a zero added.	09
F	Day of the week in month	2
E	Short name of the day of the week	Tue
EEEE	Long name of the day of the week	Tuesday
a	AM/PM marker	PM
H	Hour of the day, with the first hour being 0 and the last hour being 23.	0
HH	Two-digit hour of the day, with the first hour being 0 and the last hour being 23. Numbers less than 10 have a zero added.	08

Letter	Description	Example
k	Hour of the day, with the first hour being 1 and the last hour being 24.	24
kk	Two-digit hour of the day, with the first hour being 1 and the last hour being 24. Numbers less than 10 have a zero added.	02
K	Hour hour of the morning (AM) or afternoon (PM), with 0 being the first hour and 11 being the last hour.	0
KK	Two-digit hour of the day, with the first hour being 1 and the last hour being 24. Numbers less than 10 have a zero added.	02
h	Hour of the morning (AM) or afternoon (PM), with 1 being the first hour and 12 being the last hour.	12
hh	Two-digit hour of the morning (AM) or afternoon (PM), with 1 being the first hour and 12 being the last hour. Numbers less than 10 have a zero added.	09
m	Minute of the hour	30
mm	Two-digit minutes of the hour. Numbers less than 10 have a zero added.	05
s	Second of the minute	55
ss	Two-digit second of the minute. Numbers less than 10 have a zero added.	02
S	Millisecond of the second	978
SSS	Three-digit millisecond of the second. Numbers containing fewer than three digits will have one or two zeros added to make them three digits.	978 078 008
z	Time abbreviation of the time zone name. If the time zone does not have a name, the GMT offset.	PST GMT-08:00

Letter	Description	Example
zzzz	The full time zone name. If the time zone does not have a name, the GMT offset.	Pacific Standard Time GMT-08:00
Z	The RFC 822 time zone.	-0800
X	The ISO 8601 time zone.	-08Z
XX	The ISO 8601 time zone with minutes.	-0800Z
XXX	The ISO 8601 time zone with minutes and a colon separator between hours and minutes.	-08:00Z

Number Patterns

When defining data type options for numeric data, you can create your own custom number pattern if the predefined ones do not meet your needs. A basic number pattern consists of the following elements:

• A prefix such as a currency symbol (optional)
• A pattern of numbers containing an optional grouping character (for example a comma as a thousands separator)
• A suffix (optional)

For example, this pattern:

$ ###,###.00

Would produce a number formatted like this (note the use of a thousands separator after the first three digits):

$232,998.60

Patterns for Negative Numbers

By default, negative numbers are formatted the same as positive numbers but have the negative sign added as a prefix. The character used for the number sign is based on the locale. The negative sign is "-" in most locales. For example, if you specify this number pattern:

```
0.00
```

The number negative ten would be formatted like this in most locales:

```
-10.00
```

However, if you want to define a different prefix or suffix to use for negative numbers, specify a second pattern, separating it from the first pattern with a semicolon (";"). For example:

```
0.00;(0.00)
```

In this pattern, negative numbers would be contained in parentheses:

```
(10.00)
```

Scientific Notation

If you want to format a number into scientific notation, use the character E followed by the minimum number of digits you want to include in the exponent. For example, given this pattern:

```
0.###E0
```

The number 1234 would be formatted like this:

```
1.234E3
```

In other words, 1.234×10^3.

Note the following:

- The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum.
- Negative exponents are formatted using the localized minus sign, not the prefix and suffix from the pattern.
- Scientific notation patterns cannot contain grouping separators (for example, a thousands separator).

Special Number Pattern Characters

The following characters are used to produce other characters, as opposed to being reproduced literally in the resulting number. If you want to use any of these special charters as literal characters in your number pattern's prefix or suffix, surround the special character with quotes.

Symbol	Description

0	Represents a digit in the pattern including zeros where needed to fill in the pattern. For example, the number twenty-seven when applied to this pattern:
	
	 `0000`
	
	 Would be:
	
	 `0027`

Symbol	Description

#	Represents a digit but zeros are omitted. For example, the number twenty-seven when applied to this pattern:
	
	`####`
	
	Would be:
	
	`27`

.	The decimal separator or monetary decimal separator used in the selected locale. For example, in the U.S. the dot (.) is used as the decimal separator but in France the comma (,) is used as the decimal separator.
–	The negative sign used in the selected locale. For most locals this is the minus sign (-).

,	The grouping character used in the selected locale. The appropriate character for the selected locale will be used. For example, in the U.S., the comma (,) is used as a separator.
	
	The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands. The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last one and the end of the integer is the one that is used. For example, all the following patterns produce the same result:
	
	`#,##,###,####`
	
	`######,####`
	
	`##,####,####`

E	Separates mantissa and exponent in scientific notation. You do not need to surround the E with quotes in your pattern. See **Scientific Notation** on page 37.
;	Separates positive and negative subpatterns. See **Patterns for Negative Numbers** on page 36.

%	Multiply the number by 100 and show the number as a percentage. For example, the number .35 when applied to this pattern:
	
	`##%`
	
	Would produce this result:
	
	`35%`

Symbol	Description
¤	The currency symbol for the selected locale. If doubled, the international currency symbol is used. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.

'	Used to quote special characters in a prefix or suffix. For example:
	
	`"'#'#"`
	
	Formats 123 to:
	
	`"#123"`
	
	To create a single quote itself, use two in a row:
	
	`"# o''clock"`

Changing a Field's Data Type

Spectrum™ Technology Platform automatically changes field data types as needed using the type conversion settings specified in Management Console, or the dataflow's type conversion options specified in Enterprise Designer. In most situations you do not need to manually change field data types because any necessary data type conversions are handled automatically. However, in cases where a stage is unable to convert incoming data to the necessary data type, you may need to manually change the data type in the upstream channel.

There are only a few possible type conversions that you can perform manually. Those are:

• Polygon and MultiPolygon types can be converted to and from a geometry type.
• Date, time, and datetime data types can be converted to and from a string type.

To manually change a field's data type, follow this procedure.

1. In Enterprise Designer, double-click the channel where you want to change the field's data type. A channel is the line that connects two stages on the canvas.
2. Click the small square button next to the data type that you want to change.

    Note: If a small square button is not visible next to the data type, then manual data type conversion is not available for your situation.

3. For date, time, and datetime data types, do the following:

    Note: Only the appropriate options will be displayed depending on the data type chosen.

    a) In the **Locale** field, select the country whose format you want to use for dates converted to a string. Your selection will determine the default values in the **Date**, **Time**, and **DateTime** fields. Your selection will also determine the language used when a month is spelled out.

For example, if you specify English the first month of the year would be "January" but if you specify French it would be "Janvier."

b) In the **Date** field, select the format to use for date data when it is converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **M/D/YY** and a date field contains 2012-3-2, that date data would be converted to the string `3/2/12`.

c) In the **Time** field, select the format to use for time data when it is converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **h:mm a** and a time field contains 23:00, that time data would be converted to the string `11:00 PM`.

d) In the **DateTime** field, select the format to use for fields containing the DateTime data type when converted to a string. A list of the most commonly used formats for the selected locale is provided.

For example, if you choose the format **M/d/yy h:mm a** and a DateTime field contains 2012-3-2 23:00, that DateTime data would be converted to the string `3/2/12 11:00 PM`.

e) In the **Whole numbers** field, select the formatting you want to use for whole numbers (data types float and double).

For example, if you choose the format **#,###** then the number 4324 would be formatted as `4,324`.

> **Note:** If you leave this field blank, numbers will be formatted in the same way they were in Spectrum™ Technology Platform 8.0 and earlier. Specifically, no thousands separator is used, the dot (".") is used as the decimal separator, numbers less than 10^{-3} or greater than or equal to 10^7 are shown in scientific notation, and negative numbers have a minus sign ("-") in front of them. Also note that if you leave this field blank, numbers that use the bigdecimal data type will always be in the format #,###.000.

f) In the **Decimal numbers** field, select the formatting you want to use for numbers that contain a decimal value (data types integer and long).

For example, if you choose the format **#,##0.0#** then the number 4324.25 would be formatted as `4,324.25`.

> **Note:** If you leave this field blank, numbers will be formatted in the same way they were in Spectrum™ Technology Platform 8.0 and earlier. Specifically, no thousands separator is used, the dot (".") is used as the decimal separator, numbers less than 10^{-3} or greater than or equal to 10^7 are shown in scientific notation, and negative numbers have a minus sign ("-") in front of them. Also note that if you leave this field blank, numbers that use the bigdecimal data type will always be in the format #,###.000.

4. Click **OK**.

The color of the data type name changes to green.

5. Click **OK** again to save the change.

Changing a Field's Name

There are a variety of situations where you may need to rename a field in a dataflow. For example:

- A stage's input requires certain field names but the previous stage's output uses other field names
- There is data in a field which you want to preserve when a downstream stage writes data to a field of the same name

> **Note:** After a field is renamed, it is no longer available in subsequent stages with the old name.

1. In a dataflow, double-click the channel between two stages. The **Field Transform Options** dialog box appears.
2. Change the field name(s) as desired.

   For example, the latter stage could require "AddressLine3" but the former stage uses "FirmName" instead. In this case, you would click the drop-down arrow in the Input Field Name that corresponds to AddressLine3 as the Output Field Name and then select "FirmName."

   The color of the output field name changes to green.

3. Click **OK**.

Reserved Field Names

The following field names are used by the system so you should not use them in a dataflow.

    Status
    Status.Code
    Status.Description

Control Stages

Use control stages to move data along different paths in a flow, to split or group records, and to perform basic data transforms and mathematical operations.

Aggregator

Aggregator converts flat data to hierarchical data. It takes input data from a single source, creates a schema (a structured hierarchy of data) by grouping the data based on fields you specify, and then constructs the groups in the schema.

> **Note:** If your data includes a field by which you will group your data, such as an ID field, you must sort your data before running it through an Aggregator. You can do this by sorting the data prior to bringing it into the dataflow, by sorting the input file within Enterprise Designer (for jobs or subflows, but not services) or by adding a Sorter stage to your dataflow (for jobs, services, or subflows).

Group By

Choose the field you want to use as the basis for aggregating into a hierarchy by selecting **Group by** in the tree then clicking **Add**. Records that have the same value in the field you choose will have their data aggregated into a single hierarchy. If you select multiple fields then the data from all fields must match in order for the records to be grouped into a hierarchy.

For example, if you want to group data by account number you would select the account number field. All incoming records that have the same value in the account number field would have their data grouped into a single hierarchical record.

> **Note:** You must connect a stage to the Aggregator input port in order for a list of fields to be available to choose from.

Output Lists

The fields you choose under **Output lists** determine which fields are included in each record created by Aggregator. To add a field, select **Output lists** then click **Add** and choose one of the following options:

Existing field	Select this option if you want to add a field from the dataflow to the hierarchy.
New data type	Select this option if you want to create a parent field to which you can then add child fields.
Template	This option allows you to add a field based on data in the stage connected to the Aggregator's output port.

If you want the field to have child fields, check the **List** box.

Enter the name of the field in the **Name** text box, or leave it as-is if it auto-filled and you are satisfied with the name. Keep in mind that the Aggregator stage does not allow invalid XML characters in field names; it does allow alphanumeric characters, periods (.), underscores (_), and hyphens (-).

Click **Add** to add the field. You can specify another field to add to the same level in the hierarchy or you can click **Close**.

To add child fields to an existing field, select the parent field then click **Add**.

> **Note:** You can modify the field group by highlighting a row and clicking **Modify**, and you can remove a field group by highlighting a row and clicking **Remove**. You can also change the order of fields by clicking a field and clicking **Move Up** or **Move Down**.

Example of Aggregator

An example of the Aggregator's function is to take a group of street addresses and turn them into driving directions. You could do this with two points, such as a start point and an end point, or you could do this with multiple points along a route. The dataflow for this type of function might look like the following:
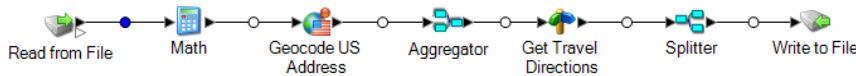
The dataflow performs the function as follows:

1. The **Read from File** stage contains street addresses in a flat file. The fields in this file include the following:

   • an **ID**, which identifies a particular address in the file
   • a **Type**, which indicates whether the address is a "From" address or a "To" address
   • an **AddressLine1** field, which provides the street address
   • a **LastLine** field, which includes such information as a city, state, and/or postal code

2. The **Field Transform** between the Read from File stage and the Math stage changes the format of the ID field from string to double because the Math stage does not accept string data.
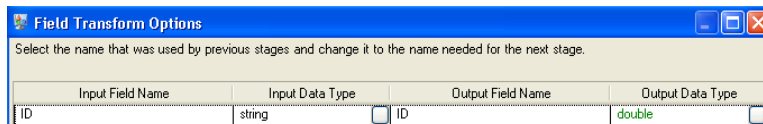
3. The **Math** stage creates an expression that establishes a Group ID field to be used downstream in the dataflow. In this example, it calculates the Group ID as the floor of, or rounds down, the value of the ID field divided by 2. So, if the ID is 3, then the expression is 3/2, which equals 1.5. When you round down 1.5, it becomes 1. If the ID is 2, then the expression is 2/2, which equals 1, and there is no need to round down. Therefore, IDs 2 and 3 have the same Group ID of 1.
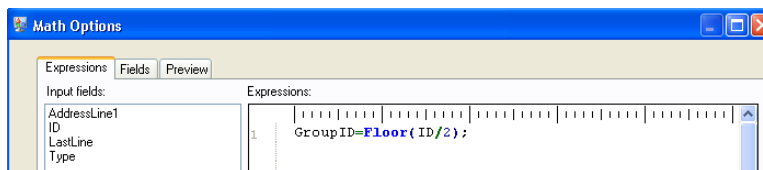
4. **Geocode US Address** obtains latitudes and longitudes for each address.

---

5. The **Aggregator** stage establishes that the data should be grouped by the GroupID field and that the output lists should include Route Points devised of latitudes and longitudes.

   The instructions below show how to **manually** configure the Aggregator stage for this dataflow:

   - Double-click the Aggregator stage, and then double-click **Group by**.
   - Select the **GroupID** field and click **OK**. Using this field will allow us to include route points for the next stage in the dataflow. Route points are essential for a dataflow that produces directions.
   - Double-click **Output lists**. The **Field Options** dialog box appears.
   - Select **New data type**. In the **Type name** field enter `RoutePoint`. In the **Name** field enter `RoutePoints`. By default, this is a list and cannot be changed, so the checkbox is grayed out.
   - Press **OK**.
   - Click **RoutePoints** and click **Add**. The **Field Options** dialog box appears again.
   - Route Points are made up of latitudes and longitudes, so we need to first add an **Existing field** from the existing input field `Latitude`. The **Name** field will auto-populate.
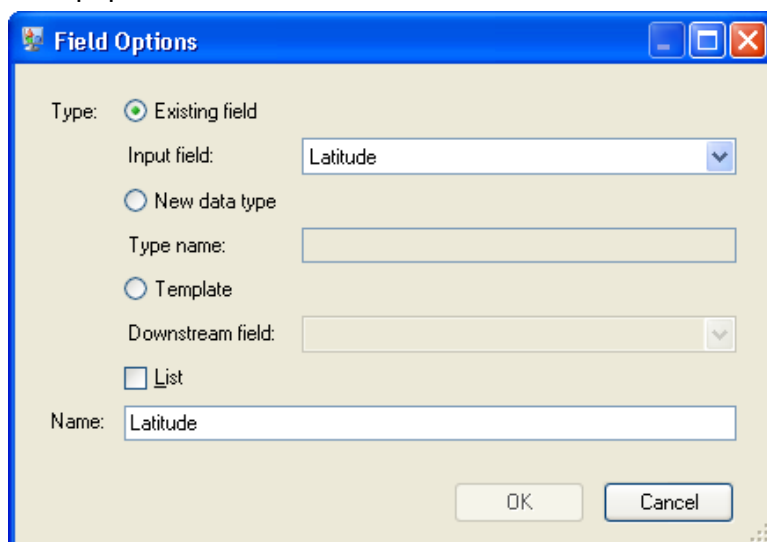
   Repeat this step for Longitude.
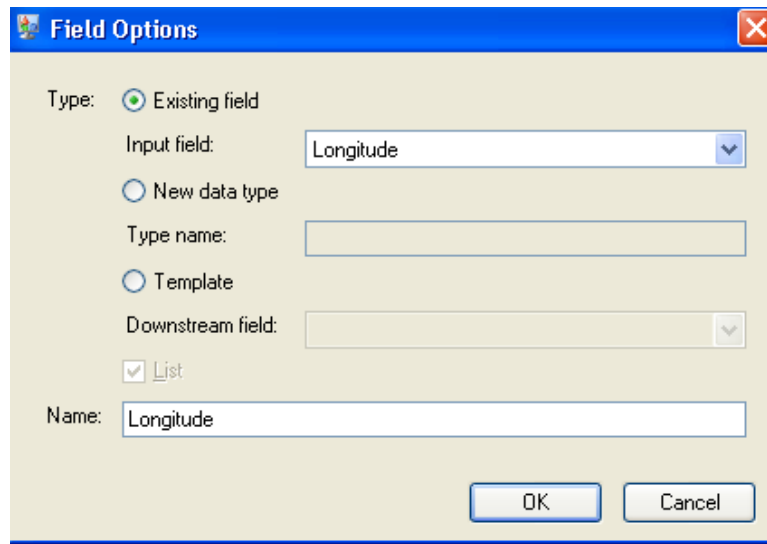
The completed Aggregator stage will appear as follows:
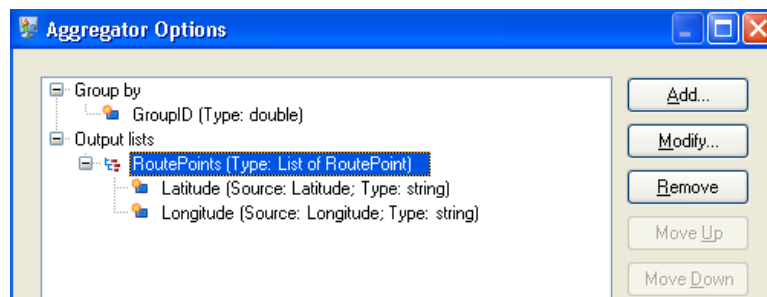
6. **Get Travel Directions** provides driving instructions from point IDs 0, 2, and 4 to point IDs 1, 3, and 5, respectively.

7. The **Splitter** stage establishes that the data should be split at the Route Directions field and that the output lists should include all of the possible fields from the Get Travel Directions stage.

8. The **Write to File** stage writes the directions to an output file.

Broadcaster

A Broadcaster takes a stream of records and splits it into multiple streams, allowing you to send records to multiple stages for simultaneous processing.

Broadcaster has no settings to change.

Conditional Router

The **Conditional Router** stage sends records to different paths in the dataflow depending on the criteria you specify. The stage can have one or more output ports, depending on the defined criteria. Output ports are numbered consecutively, starting with 1 (which displays as "port").

The output ports connect to different stages to which the data is to be sent, depending on defined conditions. For example, you can send one set of records to port 1 in case of a successful match, while a different set of records can be sent to port 2 in case of a failed match.

An input record is written to the Conditional Router's output port only if the entire expression evaluates to true.

Configuring a Conditional Router

1. Under **Control Stages**, click on **Conditional Router** and drag it onto the canvas, placing it in the desired location within the dataflow.
2. Connect the router to other stages on the canvas.

   > **Note:** This is a mandatory step before defining the port settings. Otherwise the ports are not available for editing.

3. Double-click on the **Conditional Router** stage on the canvas.
   The **Conditional Router Options** window appears.
4. Click the square button in the **Condition/Expression** column against the **port** row.
   The **Expressions Editor** window appears.
5. In the **Choose Expression Type** section, select one of the following:

   - **Expression created with Expression Builder**: Select this option to create a basic expression, where you can add Groups and Expressions, which can be combined using different logical operators. For more information, see **Using the Expression Builder** on page 46.
   - **Custom expression**: Select this option to write an expression using the Groovy scripting language. For more information, see **Writing a Custom Expression** on page 50.
   - **Default expression**: Select this to route records to this port by default. Records that do not match any of the other ports' expressions will be routed to this port. You should always have an output port with "default" as the expression to ensure that no rows are missed in case of a port mismatch, and all rows are written from the router.

6. Click **OK**. The **Expressions Editor** window closes.
7. Click **OK** on the **Conditional Router Options** window.

Using the Expression Builder

The **Expression Builder** of the **Conditional Router** stage allows you to create an expression that must evaluate to true for an input record to be routed to the output port of the stage.

1. Each parent group comprises of a desired conditional combination of child expressions and child groups.
2. Each expression consists of a left operand, a right operand and a logical operator.
3. Each group must specify whether *all* or *any* of its constituent conditions must hold true for the entire group to evaluate to true.

To build an expression using the Expression Builder:

1. In the **Expression Editor**, select the option **Expression created with Expression Builder**.

   By default, the Expression Builder option is selected and a parent group is displayed in the expression hierarchy tree on the left of the **Expression Builder** section.

2. To add a child group within the selected group, click **Add Group** .

   This newly added group gets added as a child of the parent group, and is selected in the tree by default. Within each group, you can add child expressions and child groups.

3. For each group, select either **All true** or **Any true** under the **Combine expression method** header.

   - `All true`: The group evaluates to true only if all the child criteria of the group hold true.
   - `Any true`: The group evaluates to true if even one of its child criteria hold true.

4. To add a child expression within the selected group, click **Add Expression**.

   The newly added expression gets added as a child of the parent group and is selected in the tree by default.

   To define this child expression:

   a) Specify the left operand of the selected expression using the **Field** dropdown to select any one of the columns in the input file.
   b) Specify the logical operator connecting the two components of the selected expression by selecting the appropriate operator from the **Operator** field as explained below:

   Table 1: Expression Builder Operators

Operator	Description
Is Equal	Checks if the value in the field matches the value or field specified.
Is Not Equal	Checks if the value in the field does not match the value or field specified.
Is Null	Checks if the field is a null value.
Is Not Null	Checks if the field is not a null value.

Operator	Description
Is Empty	Checks if the field is null or a string with a length of 0. **Note:** This operation is only available for fields with a data type of string.
Is Not Empty	Checks if the field is neither null nor a string with a length of 0. **Note:** This operation is only available for fields with a data type of string.
Is Less Than	Checks if the field has a numeric value that is less than the value specified. This operator works on numeric data types as well as string fields that contain numbers. **Note:** This operation is not available for fields with a data type of Boolean.
Is Less Than Or Equal To	Checks if the field has a numeric value that is less than or equal to the value specified. This operator works on numeric data types as well as string fields that contain numbers. **Note:** This operation is not available for fields with a data type of Boolean.
Is Greater Than	Checks if the field has a numeric value that is greater than the value specified. This operator works on numeric data types as well as string fields that contain numbers. **Note:** This operation is not available for fields with a data type of Boolean.
Is Greater Than Or Equal To	Checks if the field has a numeric value that is greater than or equal to the value specified. This operator works on numeric data types as well as string fields that contain numbers. **Note:** This operation is not available for fields with a data type of Boolean.
Starts With	Checks if the field begins with the characters specified. **Note:** This operation is only available for fields with a data type of string.

Operator	Description
Does Not Start With	Checks if the field does not begin with the characters specified. **Note:** This operation is only available for fields with a data type of string.
Contains	Checks if the field contains the string specified. **Note:** This operation is only available for fields with a data type of string.
Does Not Contain	Checks if the field does not contain the string specified. **Note:** This operation is only available for fields with a data type of string.
Ends With	Checks if the field ends with the characters specified. **Note:** This operation is only available for fields with a data type of string.
Does Not End With	Checks if the field ends with the characters specified. **Note:** This operation is only available for fields with a data type of string.
Matches Regular Expression	Matches the field with a regular expression for identifying strings of text of interest, such as particular characters, words, or patterns of characters. The value field should contain a valid regular expression pattern. **Note:** This operation is only available for fields with a data type of string.

c) Specify the right operand of the selected expression by selecting either `Value` or `Field`.

- `Value`: The left operand of the selected expression is compared to this value.
- `Field`: The left operand of the selected expression is compared to this column of the same input file. Select the right operand column from the dropdown.

5. To add a sibling expression or sibling group to any entity, select that entity in the tree and click **Add Expression** or **Add Group** respectively.

6. To shift a child expression or child group from one parent group to a different parent group, drag it to the desired parent group header in the criteria tree on the left.

7. Repeat the above steps to add as many child expressions and child groups as are required to create the desired final expression criteria.

8. Click **OK**.

The **Condition/Expression** column in the **Conditional Router Options** window displays the defined expression critera, which must evaluate to true for a record to be written to the stage's corresponding output port.

Writing a Custom Expression

You can write your own custom expressions to control how Conditional Router routes records using the Groovy scripting language to create an expression.

Using Groovy Scripting

For information about Groovy, see **groovy-lang.org**.

Groovy expressions used in the Conditional Router stage must evaluate to a Boolean value (true or false) which indicates whether the record should be written to the port. The record is routed to the first output port whose expression evaluates to true.

For example, if you need to route records with a validation confidence level of >=85 to one stage and records with a validation confidence level of <85 to another stage, your script would look like:

```
data['Confidence']>=85
```

The script for the other port would look like:

```
data['Confidence']<85
```

The router would evaluate the value of the Confidence field against your criteria to determine which output port to send it to.

Checking a Field for a Single Value

This example evaluates to true if the Status field has 'F' in it. This would have to be an exact match, so 'f' would not evaluate to true.

```
return data['Status'] == 'F';
```

Checking a Field for Multiple Values

This example evaluates to true if the Status field has 'F' or 'f' in it.

```
boolean returnValue = false;
if (data['Status'] == 'F' || data['Status'] == 'f')
{
 returnValue = true;
}
return returnValue;
```

Evaluating Field Length

This example evaluates to true if the PostalCode field has more than 5 characters.

```
return data['PostalCode'].length() > 5;
```

Checking for a Character Within a Field Value

This example evaluates to true if the PostalCode field has a dash in it.

```
boolean returnValue = false;
if (data['PostalCode'].indexOf('-') != -1)
{
 returnValue = true;
}
return returnValue;
```

Scripting Guidelines

1. Column names must be enclosed within either single or double quotes.

   For example, this syntax is incorrect because rhe column name `PostalCode` is not enclosed within either single or double quotes.

   ```
   return data[PostalCode];
   ```

2. A column name must be specified.

   For example, this syntax is incorrect because no column is specified.

   ```
   return data[];
   ```

3. A `return` statement must return a `boolean` value.

   For example, this script is incorrect because `row.set('PostalCode', '88989')` does not return a `boolean` value. It just sets the value of the `PostalCode` field to `88989`.

   ```
   return row.set('PostalCode', '88989');
   ```

4. Use a single equals sign (=) to set the value of a field, and a double equals sign (==) to check the value of a field.

Group Statistics

The Group Statistics stage allows you to run statistical operations across multiple data rows broken down into groups that you want to analyze. If no groups are defined all rows will be treated as belonging to one group.

Groups are defined by one or more fields that have the same value across multiple data rows.

For example, the data in this table could be grouped by region, state, or both.

Region	State
East	MD
East	MD
East	CT
West	CA
West	CA

A group by Region would yield East and West. A group by State would yield California, Connecticut, and Maryland. A group by Region and State would yield East/Maryland, East/Connecticut, and West/California.

Input

The Group Statistics stage takes any field as input. Grouping can be performed on numeric or string data.

Options

Table 2: Operations Tab

Option	Description
Input fields	Lists the fields in the dataflow which you can use to group records and perform calculations.
Row	Specifies the field or fields you want to use as categories for the calculations. For example, if you had data that included a Region field and you wanted to calculate total population by region, you would group by the Region field. To add a field, select the field in the **Input fields** list then click **>>**.

Option	Description
Column	Optional. For creating a pivot table, specifies the field or fields whose values you want to pivot into columns for the purposes of cross tabulation.
	To add a field, select the field in the **Input fields** list then click **>>**.
	For example, if you had data that includs regions and shipping dates, and you want to tally the number of shipments per day for each state, you must specify the state field as a row and the shipment date field as a column.
Rows and Columns are pre-sorted in the configured order	Indicates that the input data is already sorted.
	If this checkbox is checked, the stage does not sort the data and performs the specified operation directly on the input data.
Operation	Specifies the calculation to perform on each group. To add an operation, select the field in the **Input fields** list that you want to use for the operation then click **>>**.
	For more information about the supported Group Statistics operations, see **Operations** on page 54.
Type	For the input and output fields, specifies the data type.

Type (continued):

Integer	A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647)
Long	A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807)
Float	A numeric data type that contains both negative and positive single precision numbers between 2^{-149} (1.4E-45) and $(2-2^{23}) \times 2^{127}$ (3.4028235E38)
Double	A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} (4.9E-324) and $(2-2-52) \times 2^{1023}$ (1.7976931348623157E308)

Note: When using the integer and long types, data can be lost if the input number or calculated number from an operation contains decimal data.

Option	Description
Get count of records that are computed upon	Returns the actual number of records in a group on which the selected operation is performed.
	This column *Computational Count* excludes those input records where the column on which the operation is performed contains `null` values.

Fields Tab

The Fields tab is used when creating a pivot table. For more information, see **Creating a Pivot Table** on page 61.

Output Tab

Option	Description
Return one row per group	For each group of rows, return a single row that contains the aggregated data for all rows in the group. Individual rows will be dropped. If this option is not selected, all rows will be returned. No data will be dropped. This option is not available if you use the Percent Rank or ZScore operations.
Return a count of rows in each group	Returns the number of rows in each group. The default output field name that will contain the count is GroupCount.
Return a unique ID for each group	Returns a unique ID for each group of rows. The ID starts at 1 and increments by 1 for each additional group found. The default field name is GroupID.

Operations

The calculations available are:

Average   For each group, calculates the average value of a given field. For example, if you had a group of records with values 10, 12, 1, and 600 in a given field, the average value of that field for that group would be 155.75, calculated as (10+12+1+600)÷4.

Maximum   For each group, returns the largest value in a given field. For example, if you had a group of records with values 10, 12, 1, and 600 in a given field, the maximum value of that field for that group would be 600.

Minimum   For each group, returns the smallest value in a given field. For example, if you had a group of records with values 10, 12, 1, and 600 in a given field, the minimum value of that field for that group would be 1.

Percent Rank   For each record within a group, calculates the percentile rank of a value in a given field relative to other records in the group. The percentile rank represents the percentage of records in the group with lower values in the field.

Percentile   For each group, calculates the value that would represent the percentile you specify (0 - 100) for a given field. A percentile represents the percentage of records that have a lower score. For example, if you have a group of records with values 22, 26, and 74, and you perform a percentile calculation specifying the 60th percentile, the operation would return 35.6. This means that a record with

a value of 35.6 in the given field would be in the 60th percentile of records in the group.

Standard Deviation	For each group, calculates the standard deviation for a given field. The standard deviation measures the amount of dispersion within the group. The lower the standard deviation, the more the values are centered around the mean value, and therefore the less dispersed the values. The higher the value, the more widely dispersed the values. The standard deviation is expressed in the same units as the data. The standard deviation is the square root of the variance.
Sum	For each group, calculates the sum of the values for a given field.
Variance	For each group, calculates the variance for a given field. The variance measures the amount of dispersion within the group. It is the square of the standard deviation.
ZScore	For each record in a group, returns the ZScore. The ZScore indicates how many standard deviations a value is above or below the group's mean.
Alphabetical First	For each group, returns first dictionary value. If there are more than one field values having same length or dictionary position, it returns the first occurrence of that value. For example, if group has a record with values Joel and Joey in a field, then the alphabetical first value for a group will be Joel as l comes before y in alphabet.
Alphabetical Last	For each group, returns last dictionary value. If there are more than one field values having same length or dictionary position, it returns the last occurrence of that value. For example, if group has a record with values Joel and Joey in a field, then the alphabetical last value for a group will be Joey as y comes after l in alphabet.
Longest	For each group, returns longest value. For example, if group has a record with values Joel and Jacob in a field, then the longest length value for a group will be Jacob as it has 5 alphabets whereas Joel has 4.
Shortest	For each group, returns shortest value. For example, if group has a record with values Joel and Jacob in a field, then the shortest length value for a group will be Joel as it has 4 alphabets whereas Jacob has 5.
Latest	For each group, returns the latest date/datetime value. For example, if a group has a record with values 15-12-2014 and 24-12-2014 in a field, then the latest value for the group will be 24-12-2014.
Earliest	For each group, returns the earliest date/datetime value. For example, if a group has a record with values 15-12-2014 and 24-12-2014 in a field, then the earliest value for the group will be 15-12-2014.

Output Columns

Field Name	Description / Valid Values
<Operation>Of<InputFieldName>	Contains the result of a calculation. Group Statistics creates one output field per operation and names the field based on the operation and field. For example, the default field name for a `Sum` operation performed on a field named `Population` would be `SumOfPopulation`.
<Value>_<Operation>	Contains the result of a pivot, where `<Value>` is one of the values in a pivot column and `<Operation>` is the operation performed on the column. For more information, see **Creating a Pivot Table** on page 61.
GroupCount	Indicates the number of records in the group.
GroupID	A unique number assigned to each group sequentially. The first group has a GroupID value of 1, the second has a value of 2, and so on.
ComputationalCount<Operation> Of<InputFieldName>	Indicates the actual number of records in a group on which the operation is performed. For example, for the operation `Average` performed on the `Salary` column, the column `ComputationalCountAverageOfSalary` is generated.

Status	Reports the success or failure of the Group Statistics calculations.	
	null	Success
	F	Failure

Status.Code	Reason for failure, if there is one. The status codes available are:	
	UnableToDoGroupStatistics	The Group Statistics stage was unable to perform its calculations.
	Error calculating percentile value	The percentile value could not be calculated using the input data provided.

IN

Field Name	Description / Valid Values
Status.Description	A verbose description of the error.

The input field value could not be converted to the field type. It might be overflow!	A number in an input field is larger than the data type allows. Try converting to a data type that supports larger numbers, such as double.

Group Statistics Example

This input data shows the number of customers you have in certain counties. The data also shows the U.S. state in which the county is located (MD, VA, CA, and NV), as well as the region (East or West). The first row is a header record.

```
Region|State|County|Customers
East|MD|Calvert|25
East|MD|Calvert|30
East|MD|Prince Georges|30
East|MD|Montgomery|20
East|MD|Baltimore|25
East|VA|Fairfax|45
East|VA|Clarke|35
West|CA|Alameda|74
West|CA|Los Angeles|26
West|NV|Washoe|22
```

If you wanted to calculate the total number of customers for each region, you would define the Region field as a row in the **Operations** tab. For the operation, you would perform a sum operation on the Customers field.
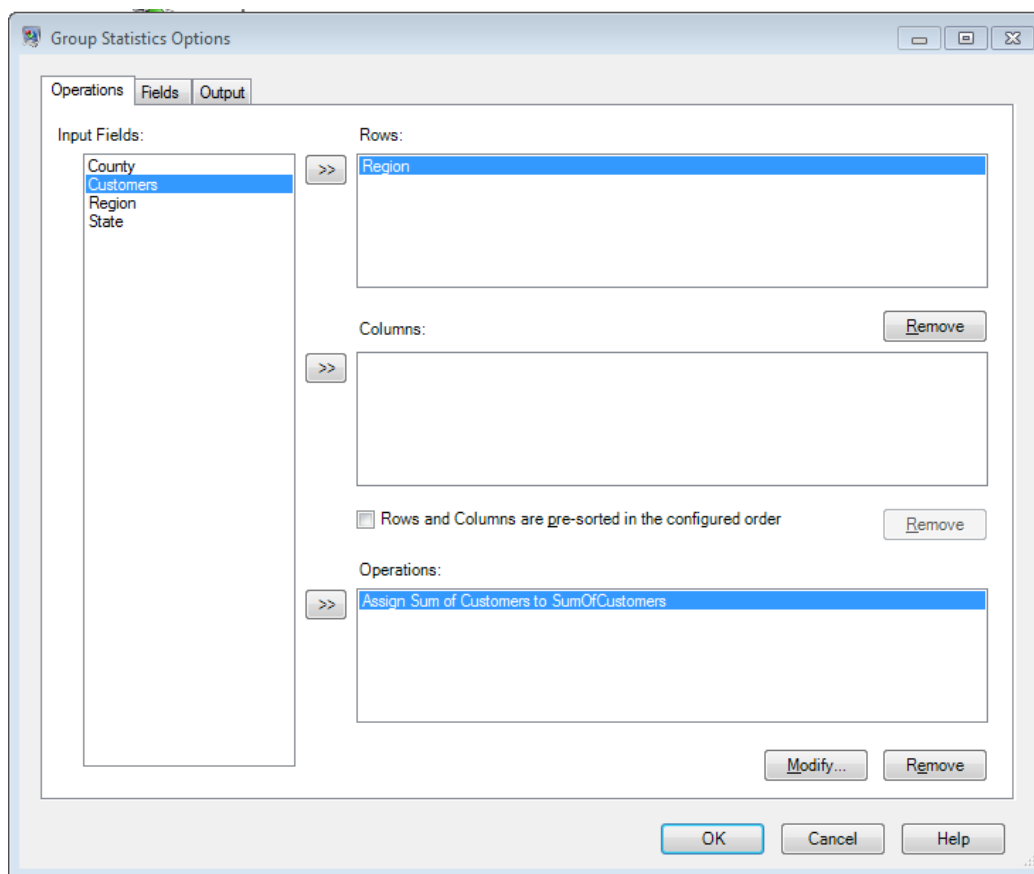
The result:

```
Region|SumOfCustomers
    East|210.0
    West|122.0
```

> **Note:** This example shows a basic group statistics operation using only rows to aggregate data. You can also create a pivot table, which aggregates both rows and columns, by specifying a column to group by in the **Operations** tab.
>
> For more information about creating a pivot table, see **Creating a Pivot Table** on page 61.

Pivot Tables

A pivot table aggregates and transposes column values in the dataflow to make it easier to analyze data visually. With pivot, you can arrange input columns into a cross tabulation format (also known as crosstab) that produces rows, columns and summarized values. You can also use fields as input

and not display them. You can use pivot to pivot on two dimensions or to group aggregate data on one dimension.

This example shows sales data for shirts.

Table 3: Input Data

Region	Gender	Style	Ship Date	Units	Price	Cost
East	Boy	Tee	1/31/2010	12	11.04	10.42
East	Boy	Golf	6/31/2010	12	13.00	10.60
East	Boy	Fancy	2/25/2010	12	11.96	11.74
East	Girl	Tee	1/31/2010	10	11.27	10.56
East	Girl	Golf	6/31/2010	10	12.12	11.95
East	Girl	Fancy	1/31/2010	10	13.74	13.33
West	Boy	Tee	1/31/2010	11	11.44	10.94
West	Boy	Golf	2/25/2010	11	12.63	11.73
West	Boy	Fancy	2/25/2010	11	12.06	10.51
West	Girl	Tee	2/25/2010	15	13.42	13.29
West	Girl	Golf	6/31/2010	15	11.48	10.67
North	Boy	Tee	2/25/2010	17	16.04	10.42
North	Boy	Fancy	2/25/2010	12	11.56	12.42
North	Girl	Tee	2/25/2010	16	12.32	18.42
North	Boy	Golf	1/31/2010	18	11.78	13.23
North	Girl	Tee	2/25/2010	12	18.45	11.64

Region	Gender	Style	Ship Date	Units	Price	Cost
North	Girl	Golf	2/25/2010	14	11.23	19.85
North	Boy	Fancy	1/31/2010	16	12.54	13.42
North	Girl	Tee	2/25/2010	17	181.73	15.83
South	Boy	Fancy	1/31/2010	19	14.15	13.42
South	Girl	Tee	2/25/2010	11	11.85	12.92
South	Girl	Fancy	1/31/2010	13	11.54	14.35
South	Boy	Tee	2/25/2010	15	14.14	14.73
South	Boy	Golf	2/25/2010	16	17.83	17.83
South	Girl	Fancy	6/31/2010	11	18.24	12.35
South	Girl	Tee	1/31/2010	20	19.94	12.95
South	Boy	Golf	2/25/2010	12	21.25	19.56

We want to be able to determine how many units we sold in each region for every ship date. To do this, we use pivot to generate this table:

Table 4: Pivot Table

Region	1/31/2010_ShipDate	2/25/2010_ShipDate	6/31/2010_ShipDate
East	32	12	22
North	34	88	
South	52	54	11
West	11	37	15

In this case, the column is Ship Date, the row is Region, and the data we would like to see is Units. The total number of units shipped is displayed here using a sum aggregation.

Creating a Pivot Table

A pivot table summarizes data for easier analysis by creating table row and column categories based on input data. For more information, see **Pivot Tables** on page 58.

In the **Group Statistics** stage options:

1. In the **Operations** tab, select a field from **Input Fields** which contains the data you want to use as the row labels in your pivot table. Then click the **>>** button next to the **Rows** field.

2. Select a field that contains the data you want to use as the columns in your pivot table then click the **>>** button next to the **Columns** field.

   Tip: At this point, run inspection to see the results of your selections. This will help you visualize the results of the cross tabulations based on the columns and rows you have selected.

3. To skip sorting the input records, check **Rows and Columns are pre-sorted in the configured order**.

   If this field is checked, the stage processes the input records without sorting them.

   > **Note:** Check this if the records are already sorted.

4. To define the operation to be performed, click the **>>** button next to the **Operations** field.

   In the **Add Operation** window:

   a) Select the **Operation** to be performed.
   b) In the **Input Field** section, select the **Name** and **Type** of the input field on which the operation must be performed.
   c) In the **Output Field** section, enter the **Name** and select the **Type** of the output field to be generated once the operation is performed.
   d) To fetch the actual count of input records on which the operation is performed as a separate output column, check **Get count of records that are computed upon**.

      Records with `null` values are not included in the count `ComputationalCount<Operation>Of<InputFieldName>`.

      Functions on which Computational Count is supported:

      - Average
      - Variance
      - ZScore
      - Standard Deviation
      - Percentile
      - Percent Rank
      - Sum

For any other operation, this checkbox remains disabled.

5. To define the output fields for each column in the pivot table, click the **Fields** tab of the stage options.

Tip: In order to define fields accurately, run an inspection flow once, before this step, to see the column names generated by your data.

a) Click **Add**.
The **Add Field** window appears.

b) In the **Add Field** window, the grid columns are based on the **Columns** fields you chose in the **Operations** tab. In these grid columns, enter those values that you see as the column headings on running an inspection flow.

The records in the column Data can also be populated in one go by using the Import feature. To import data from a CSV or TXT file:

1. Click **Import**
2. Browse the source file using **File name** field
3. Enter the **Field and Record Separator** values
4. Click OK

All the records in the file get populated in the Column Data table.

> **Note:** The source file should not have any header row.

For example, if you selected an input field called `ShipDat e` in **Columns** in the **Operations** tab, the grid in the **Add Field** window would have a column labeled `ShipDate`. In this grid column, enter the exact `ShipDate` values present in your dataflow's input data, such as `2/25/2010`, `1/31/2010`.

c) In the **Operation** field, select one or more operations for which output columns are generated for each entered column field value. Note that the operation you select only affects the field name and does not control the actual calculation.

To change the operations listed in the **Operation** field of the **Fields** tab, modify the **Operations** field values in the **Operations** tab.

Attention: The Computational Count operation option `ComputationalCount<Operation>Of<InputFieldName>` is listed only if the **Get count of records that are computed upon** checkbox is selected while defining the **Operation** in the **Operations** tab.

d) Click **Add**.

6. Click **OK**.

For each input value you entered in the grid above, output columns are automatically created by mapping those against each of the selected **Operation** values. A *cartesian product* of the entered input column values in the grid and the selected Operations is used to automatically generate the final output columns.

The names of these output columns follow the naming convention
`<Data>_<Operation>Of<InputFieldName>`, where `<Data>` is the value you specified in the first field, `<Operation>` is the operation you selected in the **Operation** field, and `<InputFieldName>` is input column on which the `operation` is performed.

Pivot Table Example

The input data which shows shipping information from the fulfillment department:

```
Region,State,County,ShipDate,Unit
East,MD,Calvert,1/31/2010,
East,MD,Calvert,6/31/2010,212
East,MD,Calvert,1/31/2010,633
East,MD,Calvert,6/31/2010,234
East,MD,Prince Georges,2/25/2010,112
East,MD,Montgomery,1/31/2010,120
East,MD,Baltimore,6/31/2010,210
East,VA,Fairfax,1/31/2010,710
West,CA,SanJose,1/31/2010,191
West,CA,Alameda,2/25/2010,411
West,CA,Los Angeles,2/25/2010,
West,CA,Los Angeles,2/25/2010,215
West,CA,Los Angeles,6/31/2010,615
West,CA,Los Angeles,6/31/2010,727
```

To determine the number of shipments that went out on each shipping date for each state, create a pivot table by configuring the Group Statistics stage as illustrated:
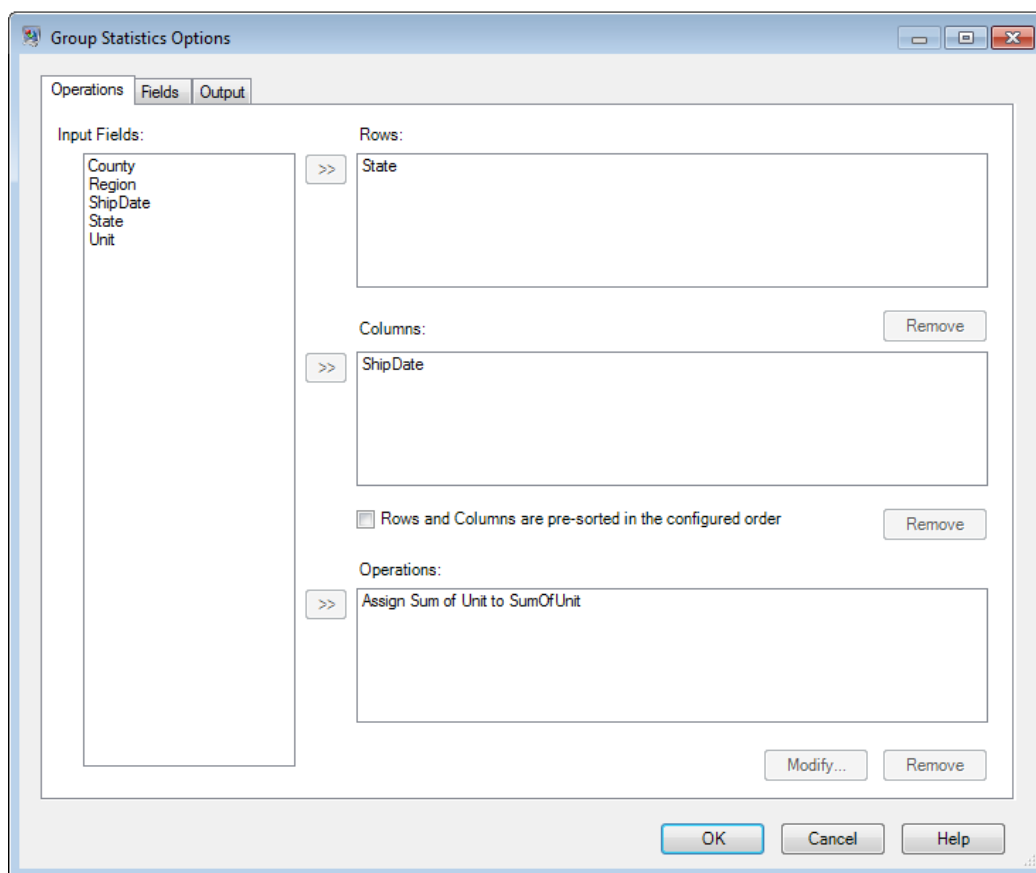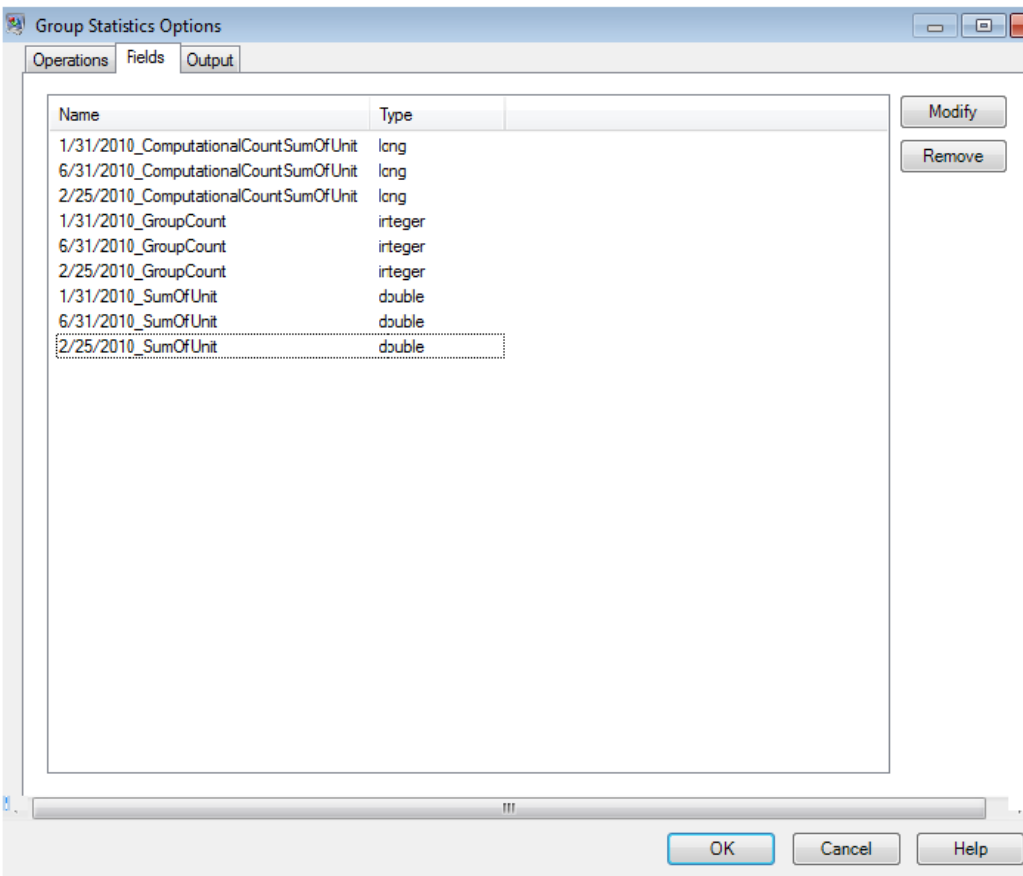
In the **Fields** tab of the stage options, add the exact dates in the grid that appear in the `ShipDate` field of the dataflow's input data, and select the **Operation** values to be displayed for each of these column values.
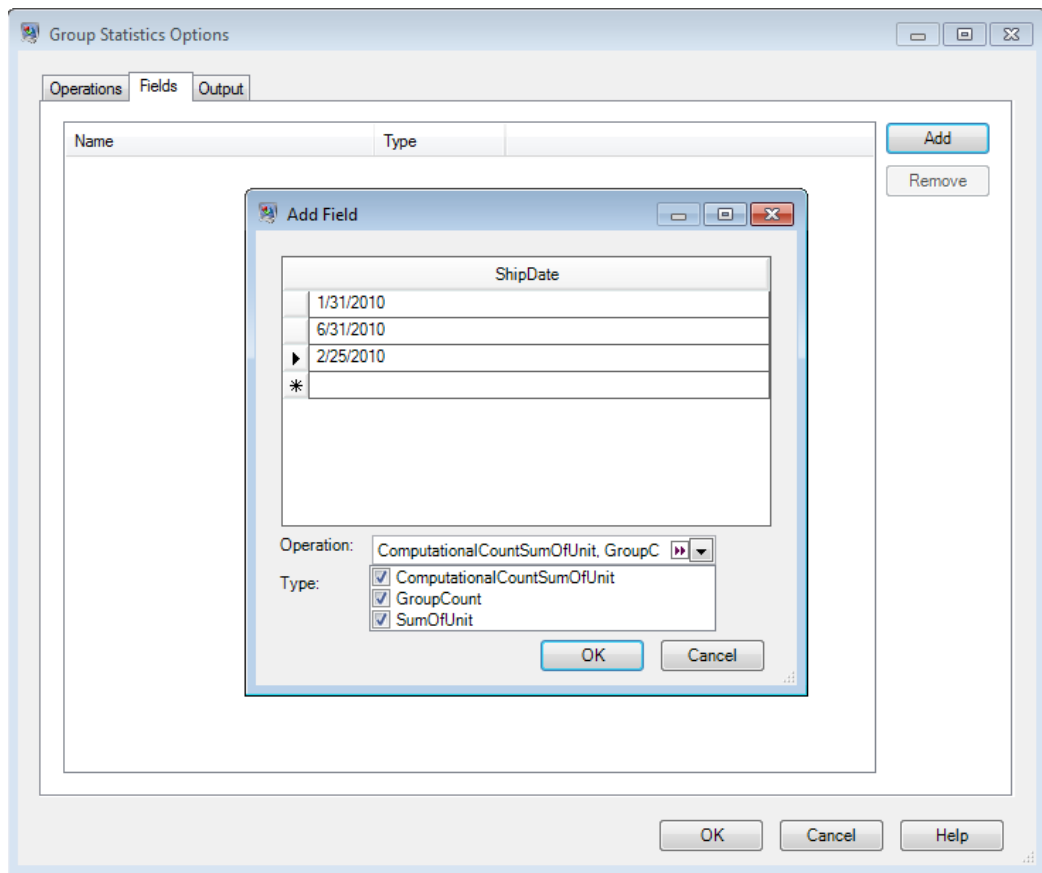
On clicking **OK** in the **Add Field** window, the output columns to be created are automatically listed in the **Fields** tab. These output columns are a *cartesian product* of the exact input values and the operations you selected in the **Add Field** window.

Output

```
State,1/31/2010_GroupCount,1/31/2010_ComputationalCountSumOfUnit,
1/31/2010_SumOfUnit,2/25/2010_GroupCount,2/25/2010_ComputationalCountSumOfUnit,
2/25/2010_SumOfUnit,6/31/2010_GroupCount,6/31/2010_ComputationalCountSumOfUnit,
6/31/2010_SumOfUnit
VA,1,1,710,,,,,,
CA,1,1,191,3,2,626,2,2,1342
MD,3,2,753,1,1,112,3,3,656
```

Math

The Math stage handles mathematical calculations on a single data row and allows you to conduct a variety of math functions using one or more expressions. Data is input as strings but the values must be numeric or Boolean, based on the type of operation being performed on the data.

1. Under Control Stages, click the Math stage and drag it to the canvas, placing it where you want on the dataflow.
2. Connect the stage to other stages on the canvas.

3. Double-click the Math stage. The **Math Options** dialog box appears, with the Expressions tab open. This view shows the input fields, the Calculator, and the Expressions canvas. Alternately, you can click the Functions tab to use functions instead of the Calculator.

The Input fields control lists the valid fields found on the input port. Field name syntax is very flexible but has some restrictions based on Groovy scripting rules. If you are not familiar with Groovy scripting, see this website for complete information on Groovy:**groovy-lang.org**.

Using the Calculator

The Calculator control contains buttons for entering numeric constants and operators into an expression. Double-clicking fields, constants, operators, and functions will insert them into an expression.

Table 5: Calculator Operators

Operator	Description
Backspace	Used to go back one space in an expression
pi	Pi, a mathematical constant which is the ratio of the circumference of a circle to its diameter
e	Euler's Number, a mathematical constant that is the base of the natural logarithm
/	Division
*	Multiplication
+	Addition
-	Subtraction
x^y	Power of (e.g., x^2 is x to the power of 2)
Mod	Modulo, the remainder of an operation
;	Semicolon, used at the end of expressions
=	Assignment operator

Operator	Description
()	Parentheses, to represent hierarchy in an expression
.	Decimal point
if\else	Conditional statement to take action if a condition is true, otherwise, take a different action
if\else if\else	Multiple conditional statement to take action if a condition is true, otherwise, take a different action
==	Equal to, in a math function
!=	Not equal to
&&	Logical and
||	Logical or
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using Functions and Constants

The Math stage provides several functions that can be used in an expression. Functions take the general form function(parameter); function(parameter,parameter); function(parameter,...), where "parameter" is a numeric constant, a variable, or a math expression. Functions can be used with other math expressions (e.g., x=Sin(y)*Cos(z)).

Constants, Conversion, Math, and Trigonometry. Each of the supported functions is listed below within its corresponding category.

Table 6: Supported Functions

Function	Description
Constants	
e	A mathematical constant that is the base of the natural algorithm.
false	A Boolean constant that represents the value false.
Infinity	A mathematical constant that represents infinity.
NaN	A mathematical constant that represents a value that is not a number.
Pi	A mathematical constant that is the ratio of the circumference of a circle to its diameter.
true	a Boolean constant that represents the value true.
Conversion	
Abs (value)	Takes one parameter. Returns the absolute value of the given value.
Ceil (value)	Takes one parameter. Returns a rounded-up value (e.g., Ceil(5.5) returns 6).
DegToRad (value)	Takes one parameter. Converts a given value from degrees to radians.
Floor (value)	Takes one parameter. Returns a rounded-down value (e.g., Floor(5.5) returns 5).
RadToDeg (value)	Takes one parameter. Converts a given value from radians to degrees.
Round (value)	Takes one parameter. Returns a rounded value.

Function	Description
Math	
Avg (value, value,...)	Takes one or more parameters. Returns the average of all given values.
Exp (value)	Takes one parameter. Returns Euler's number raised to the power of the value.
Fac (value)	Takes one parameter. Returns the factorial of a given value (e.g., Fac(6) is computed to 6*5*4*3*2*1 and returns 720).
Ln (value)	Takes one parameter. Returns the natural logarithm (base e) of a given value.
Log (value)	Takes one parameter. Returns the natural logarithm (base 10) of a given value.
Max (value, value,...)	Takes one or more parameters. Returns the maximum value passed in.
Min (value, value,...)	Takes one or more parameters. Returns the minimum value passed in.
Sqrt (value)	Takes one or more parameters. Returns the square root of the value passed in.
Sum (value)	Takes one parameter. Returns the sum of the given values.
Trigonometry	
ArcCos (value)	Takes one parameter. Returns the arc cosine of a value.

Function	Description
ArcSin (value)	Takes one parameter. Returns the arc sine of a value.
ArcTan (value)	Takes one parameter. Returns the arc tangent of a value.
Cos (value)	Takes one parameter. Returns the cosine of a value.
Ln (value)	Takes one parameter. Returns the natural logarithm (base e) of a given value.
Sin (value)	Takes one parameter. Returns the sine of a value.
Tan (value)	Takes one parameter. Returns the tangent of a value.

Using Conditional Statements

Conditional statements can be used to take actions depending on whether various conditions evaluate to true or false. Grouping using parentheses (and) can be used for more complex conditions.

Table 7: Conditions

Condition	Description
Equals	expression = = expression
Not Equals	expression != expression
Greater Than	expression > expression
Greater Than or Equal To	expression >= expression

Condition	Description
Less Than	expression < expression
Less Than or Equal To	expression <= expression
Not condition	!condition
And	condition && condition
Or	condition || condition

If Statement

```
if(condition)
{
     actions to take if condition is true
}
```

Brackets are needed only if more than one statement is executed after the "if."

If-Else If Statements

```
if(condition)
{
     actions to take if condition is true
}
else if(condition)
{
     actions to take if condition is true
}
else if...
```

```
if(SideLength != NaN)
 {
 AreaOfPolygon=
 ((SideLength^2)*NumberOfSides)/
 (4*Tan(pi/NumberOfSides));
 }
 else if(Radius != NaN)
 {
 AreaOfPolygon=
 (Radius^2)*NumberOfSides*Sin((2*pi)/NumberOfSides)/2;
 }
```

One or more else if statements can be specified. Brackets are needed only if more than one statement is executed after the "if-else- if-else."

Else-If Statement

```
if(condition)
  {
   actions to take if condition is true
  }
  else if(condition)
  {
   actions to take if condition is true
  }
  else if...
  else
  {
   actions to take if no conditions are met
  }
```

Using the Expressions Console

The Expressions console is used to enter math expressions to be evaluated by the Math stage. The Input, Calculator, and Functions controls are used to insert values into this console. You can also manually type expressions into the console. Expressions take the form of a constant, variable, or math operation, and consist of numeric constants and variables. Numeric constants are whole or decimal numbers, which can be signed. Variables represent data from the incoming row; for example, if fields x, y, and z are defined in the input, then x, y, and z can be used in an expression. Variables are replaced with field values at runtime.

The Math stage also allows grouped expressions, which involve using parentheses to group expressions and override operator precedence. For example, 2*5^2 equals 50, while (2*5)^2 equals 100.

> **Note:** Every expression you enter must end with a semi-colon.

Additionally, conditional statements can be used in the Expressions console to take actions depending on whether various conditions evaluate to true or false. See **Using Conditional Statements** on page 71 for more information on conditional statements.

The Math stage deals primarily with assignment expressions, in which the output from an expression is assigned to a variable. Multiple assignment operations are supported in the stage and can use the output of a previous assignment operation.

Assignment Expression Examples

In the scenario below, x=10 and z=1000:

```
x=5+5
z=x*100
```

In the scenario below, the area of a polygon is calculated based on the length of one side and the number of sides.

```
AreaOfPolygon=
      ((SideLength^2)*NumberOfSides)/
      (4*Tan(pi/NumberOfSides));
```

Using the Fields Control

The Fields control allows you to change input and output field types. You can change field types from within this control by clicking the drop-down arrow in the Type column and selecting from the list, which includes the following options:

boolean A logical type with two values: true and false. Boolean variables can be used in conditional statements to control flow. The following code sample shows a Boolean expression:

```
if(x && y)
    z=1;
  else if(x)
    z=2;
  else if(y)
    z=3;
  else
    z=4;
```

double A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52})\times2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.

float A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23})\times2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.

integer A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and $2^{31}-1$ (2,147,483,647).

long A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and $2^{63}-1$ (9,223,372,036,854,775,807).

Using the Preview Control

The Preview control allows you to test math expressions. Fields are listed in the Input Data area; you can provide specific values to pass to the expression and view the output in the Results area beneath Input Data.

Numeric fields are initialized to 0 (0.000 for double) and boolean fields are initialized to False. Double and float fields are limited to four decimal places, and integer and long fields have no decimal places.

Record Combiner

Record Combiner combines two or more records from multiple streams into a single record. Record Combiner can have one or more stage input ports. For example, you can have one group of records from one stage input (port) and the other group from a second stage input (port 2), and the records will merge into a single record. If you delete a middle stage, the ports will not renumber consecutively.

> **Note:** Record Combiner will not release a record on output until each of its input ports has received a record. It must combine as many records as it has input ports before outputting a record.

You can specify which port should be preserved in cases where the input streams have fields of the same name. For example, if you are combining records from two streams, and both streams contain a field named AccountNumber, you could specify which stream's AccountNumber field you want to preserve by choosing the Record Combiner input port that corresponds to the stream you want to preserve. The data from the AccountNumber field in the other stream would be discarded.

Record Joiner

Record Joiner performs a SQL-style `JOIN` operation to combine records from different streams based on a relationship between fields in the streams. You can use Record Joiner to join records from multiple files, multiple databases, or any upstream channels in the dataflow. You must connect at least two input channels to Record Joiner. The results of the `JOIN` operation are then written to one output channel. Optionally, records that do not match the join condition can be written to a separate output channel.

> **Note:** Before using Record Joiner you should have a good understanding of the SQL `JOIN` operation. For more information, see **wikipedia.org/wiki/Join_(SQL)**.

Join Definition

Option	Description
Left port	The port whose records you want to use as the left table in the `JOIN` operation. All other input ports will be used as right tables in the `JOIN` operation. > **Note:** "Left" table and "right" table are SQL `JOIN` concepts. Before using Record Joiner you should have a good understanding of the SQL `JOIN` operation. For more information, see **wikipedia.org/wiki/Join_(SQL)**.

Option	Description
Join type	The type of JOIN operation you want to perform. One of the following:

	Left Outer	Returns all records from the left port even if there are no matches between the left port and the other ports. This option returns all records from the left port plus any records that match in any of the other ports.
	Full	Returns all records from all ports.
	Inner	Returns only those records that have a match between the left port and another port. For instance, if you have four input sources and port 1 is the left port, an inner join will return records that have matching fields between port 1 and port 2, port 1 and port 3, and port 1 and port 4.

Option	Description
Join Fields	The field or fields from the left port that must match the data in a field from another port in order for the records to be joined.

> **Note:** Only fields that have a data type of string or integer or date or datetime can be used as join fields.

Option	Description
Data from the left port is sorted	Specifies whether the records in the left port are already sorted by the field specified in **Join Fields**. If the records are already sorted, checking this box can improve performance. If you do not check this box, Record Joiner will sort the records according to the field specified in **Join Fields** before performing the join operation.

If you have specified multiple join fields, then the records must be sorted using the order of the fields listed in **Join Fields**. For example, if you have two join fields:

> Amount
> Region

Then the records must be sorted first by the Amount field, then by the Region field.

Important: If you select this option but the records are not sorted, you will get incorrect results from Record Joiner. Only select this option if you are sure that the records in the left port are already sorted.

Option	Description
Join Definitions	Describes the join conditions that will be used to determine if a record from the left port should be joined with a record from one of the other ports. For example: `port1.Name = port2.Name` This indicates that if the value in the Name field of a record from port1 matches the value in the Name field of a record from port2, the two records will be joined. To modify a join condition, click **Modify**. Select a field from the right port whose data must match the data in the join field from the left port in order for the records to be joined. If you want to change the left port field, click **Cancel** and change it in the **Join Fields** field. If the records in the right port are sorted by the join field, check the box **Data from the right port is sorted**. Checking this box can improve performance. **Important:** If you select **Data from the right port is sorted** but the records are not sorted, you will get incorrect results from Record Joiner. Only select this option if you are sure that the records in the right port are already sorted.

Field Resolution

This tab specifies which port's data to use in the joined record in cases where the same field name exists in more than one input port. For example, if you are performing a join on two sources of data, and each source contains a field named DateOfBirth, you can specify which port's data to use in the DateOfBirth field in the joined record.

If there are fields of the same name but with different data, and you want to preserve both fields' data in the joined record, you must rename one of the fields before the data is sent to Record Joiner. You can use the Transformer stage to rename fields.

Handling Records That Are Not Joined

In order for a record to be included in the Record Joiner output it must meet the join condition, or a join type must be selected that returns both joined records and those that did not meet the join condition. For example, a full join will return all records from all input ports regardless of whether a record meets the join condition. In the case of a join type that does not return all records from all ports, such as a left outer join or an inner join, only records that match the join condition are included in the Record Joiner output.

To capture the records that are not included in the result of the join operation, use the **not_joined** output port. The output from this port contains all records that were not included in the regular output port. The **not_joined** output port is the white triangle on the right side of the Record Joiner stage as shown here:

Records that come out of this port have the field **InputPortIndex** added to them. This field contains the number of the Record Joiner input port where the record came from. This allows you to identify the source of the record.

> **Note:**
>
> • For optimal performance of this stage, ensure two independent streams of records are joined to generate a consolidated output.
> • If a single path is first branched using either a broadcaster or conditional router and then joined back using a Record Joiner, the flow may hang. In case multiple stages are used between branching and joining, use the Sorter as close to the Record Joiner as possible.

Sorter

Sorter sorts records using the fields you specify. For example, you can have records sorted by names, cities, or any other field in your dataflow.

Sorting Records with Sorter

The Sorter stage allows you to sort records using the fields you specify.

1. Under Control Stages, drag Sorter to the canvas, placing it where you want on the dataflow.
2. Double-click Sorter.
3. Click **Add**.
4. Click the down-arrow in the **Field Name** column and select the field that you want to sort on.

   > **Note:** The list of available fields is based on the fields used in the previous stages in the dataflow.

5. In the **Order** column, choose whether you want to sort in ascending or descending order.
6. In the **Type** column, select the field's data type.

   > **Note:** If your incoming data is not in string format, the **Type** column will be disabled.

bigdecimal	A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high

degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.

double A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52}) \times 2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.

float A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23}) \times 2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.

integer A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647).

long A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807).

string A sequence of characters.

7. To remove blank space from before and after the value before sorting, check the box in the **Trim** column. The trim option does not modify the value of the field. It only trims the value for the purpose of sorting. Note that if your incoming data is not in string format, the **Trim** column will be disabled.

8. In the **Treat Null As** column, select **Largest** or **Smallest** to indicate the placement of null values in the sorted list. The placement depends on the combination of options selected in the **Order** and **Treat Null As** fields, as shown in the table below:

Order	Treat Null As	Placement of null values in the sorted list
Ascending	Largest	Bottom of the list
Ascending	Smallest	Top of the list
Descending	Largest	Top of the list
Descending	Smallest	Bottom of the list

9. Repeat until you have added all the fields you want to sort.

10. Rearrange the sort order as desired by clicking **Up** or **Down**. This allows you to sort first by one field, then sort the resulting order again by another field.

11. If you want to override the default sort performance options that have been defined by your administrator, click **Advanced**, check the **Override sort performance options** box, then specify the following options:

In memory record limit	Specifies the maximum number of data rows a sorter will hold in memory before it starts paging to disk. By default, a sort of 10,000 records or less will be done in memory and a sort of more than 10,000 records will be performed as a disk sort. The maximum limit is 100,000 records. Typically an in-memory sort is much faster than a disk sort, so this value should be set high enough so that most of the sorts will be in-memory sorts and only large sets will be written to disk.

> **Note:** Be careful in environments where there are jobs running concurrently because increasing the **In memory record limit** setting increases the likelihood of running out of memory.

Maximum number of temporary files	Specifies the maximum number of temporary files that may be used by a sort process. Using a larger number of temporary files can result in better performance. However, the optimal number is highly dependent on the configuration of the server running Spectrum™ Technology Platform. You should experiment with different settings, observing the effect on performance of using more or fewer temporary files. To calculate the approximate number of temporary files that may be needed, use this equation:

```
(NumberOfRecords × 2) ÷ InMemoryRecordLimit =
NumberOfTempFiles
```

Note that the maximum number of temporary files cannot be more than 1,000.

Enable compression	Specifies that temporary files are compressed when they are written to disk.

> **Note:** The optimal sort performance settings depends on your server's hardware configuration. Nevertheless, the following equation generally produces good sort performance:

```
(InMemoryRecordLimit × MaxNumberOfTempFiles ÷ 2) >=
TotalNumberOfRecords
```

12. Click **OK**.

> **Note:** You can remove the sort criteria as desired by highlighting a row and clicking **Remove**.

Splitter

A Splitter converts hierarchical data to flat data. Splitters have one input port and one output port that delivers data from the Splitter to the next stage. One way you could use the Splitter's functionality is to take a list of information in a file and extract each discrete item of information into its own data row. For example, your input could include landmarks within a certain distance of a latitudinal/longitudinal point, and the Splitter could put each landmark into a separate data row.

Using the Splitter Stage

1. Under Control Stages, click the **Splitter** and drag it onto the canvas, placing it where you want on the dataflow and connecting it to input and output stages.
2. Double-click the Splitter. The **Splitter Options** dialog box appears.
3. Click the **Split at** drop-down to see other list types available for this stage. Click the list type you want the Splitter to create. The **Splitter Options** dialog box will adjust accordingly with your selection, showing the fields available for that list type.

   Alternatively, you can click the ellipses (**...**) button next to the **Split at** drop-down. The **Field Schema** dialog box appears, showing the schema for the data coming into the Splitter. The list types are shown in bold, followed by the individual lists for each type. Also shown is the format of those fields (string, double, and so on). Click the list type you want the Splitter to create and click **OK**. The **Splitter Options** dialog box will adjust accordingly with your selection, showing the fields available for that list type.

4. Click **Output header record** to return the original record with the split list extracted.
5. Click **Only when input list is empty** to return the original record only when there is no split list for that record.
6. Select which fields you want the Splitter to include on output by checking the **Include** box for those fields.
7. Click **OK**.

> **Splitter Example**
>
> The following example takes output from a routing stage that includes driving directions and puts each direction (or list item) into a data row. The dataflow looks like this:
>
>
>
> The dataflow performs the function as follows:
>
> 1. The **Read from File** stage contains latitudes, longitudes, and input key values to help you identify the individual points.
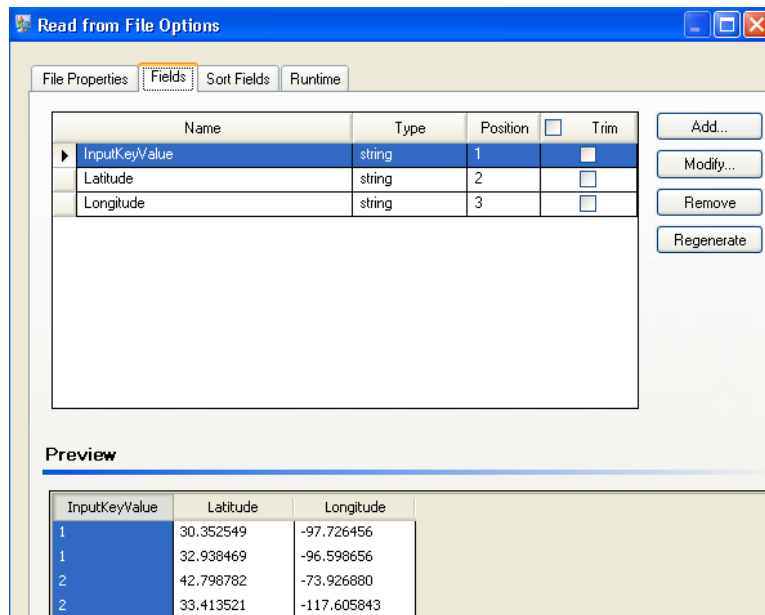
2. The Aggregator stage builds up the data from the Read from File stage into a schema (a structured hierarchy of data) and identifies the group of latitudes and longitudes as a list of route points, which is a necessary step for the next stage to work correctly.
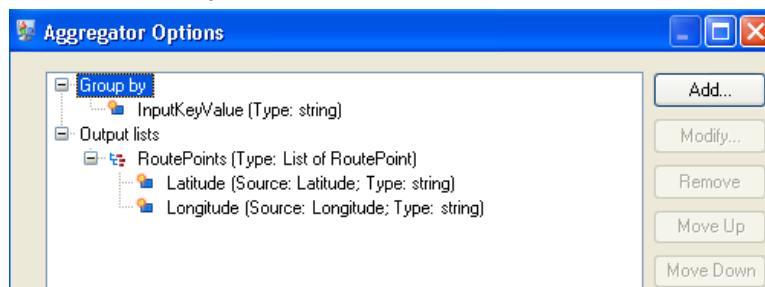
3. Location Intelligence Module's **Get Travel Directions** stage creates directions from one location to another using the route points from step 2.

4. The **Splitter** stage establishes that the data should be split at the Route Directions field and that the output lists should include all of the possible fields from the Get Travel Directions stage.
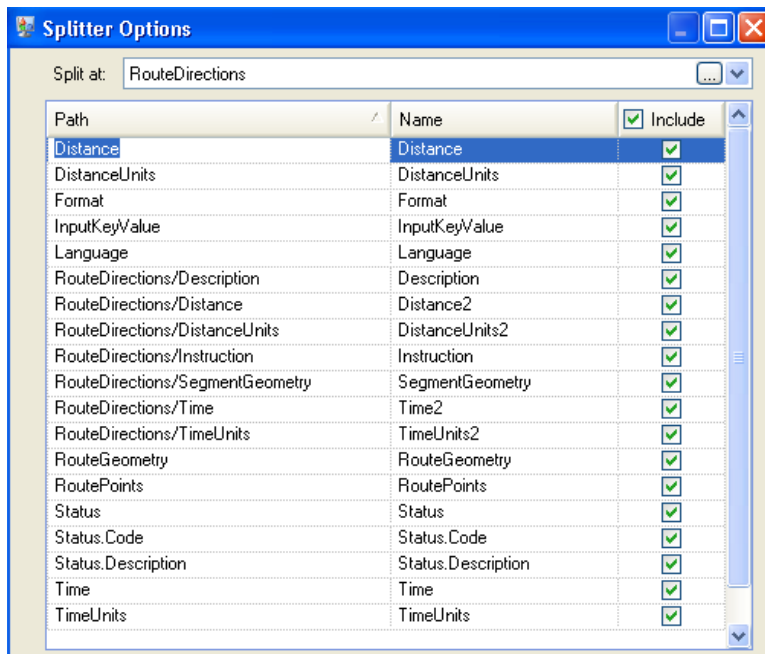
The schema is structured as follows, with Route Directions and Route Points being the available list types for this job:
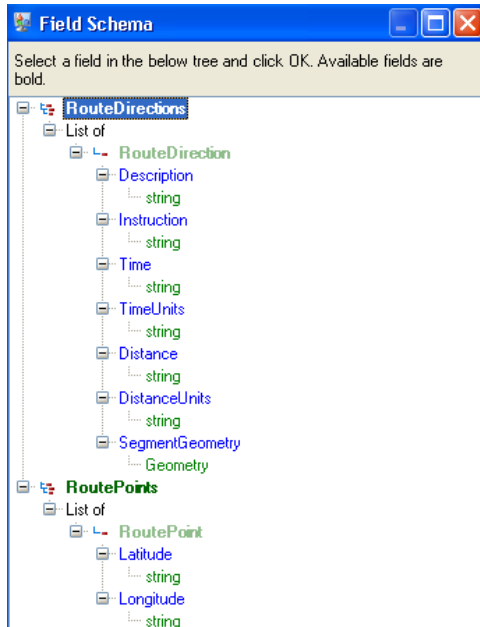
5. The **Write to File** stage writes the output to a file.

Stream Combiner

Stream Combiner joins two or more streams of records from multiple stages. Stream Combiner has one or more stage input ports. For example, you can have one group of records from one stage and another group from a second stage, and the records will merge into a single stream.

Stream Combiner has no settings.

Transformer

The Transformer stage modifies field values and formatting. You can perform more than one transform on a field as long as the input and output field names are identical.

General Transforms

Construct Field	Uses values from existing fields and/or constant values to either replace field values or create a new field. For example, say you have a field named City and you want to add the phrase "City of" to the values in the City field. You would create a template like this:

```
City of ${City}
```

In the **To field** field, you would select the City field. This has the effect of replacing the existing values in the City field with a value constructed using the template. For example, if the value in the City field is Chicago, the new value would be City of Chicago.

Some characters must be preceded by a backslash ("\") in order to produce a valid template. For example, the single quote character must be preceded by a backslash like this: `\'`. See **groovy-lang.org/syntax.html** for a list of characters that must be escaped with a backslash.

Copy	Copies the value from one field to another.
Custom	Allows you to define your own transform using the Groovy language. For more information, see **Creating a Custom Transform** on page 88.

For users of the Location Intelligence Module, custom transforms can access spatial datasets. See the Stages section in the *Spectrum Spatial Guide* on **support.pb.com**.

Rename	Changes the name of a field. You can select from a list of field names already in the dataflow or you can type the name you want.
Status	Changes the Status field to a value of either Success or Fail. When set to Fail, an optional Description and Code may also be set.

Formatting Transforms

Case	Changes casing upper or lower case.
Mask	Applies or removes characters from a field. For more information, see **Using a Mask Transform** on page 95.
Pad	Adds characters to the left or right of the field value.

String Transforms

Minimize Whitespace	Removes whitespace at the beginning and end of the field. It also replaces any sequence of whitespaces (such as multiple, consecutive spaces) to a single whitespace character.
Remove Substring	Removes all occurrences of a string from a field. For example, you could remove "CA" from the StateProvince field.
Substring	Copies a contiguous sequence of characters from one field to another.
Trim	Removes specified characters from the left, right, or both sides of a field. Note that this transform is case-sensitive.
Truncate	Removes a specified number of characters from the left and rightsides of a field.

List Transforms

This feature helps you to create canned transformation that operate on lists, for example input from read from XML.

For defining list transformations, follow these steps:

1. Select a list transformation operation. Input fields appear in a tree view on the right.
2. Select a valid field in the tree to apply the operation on. Properties for the operation show up below the input fields tree view.
3. Specify the operation properties and click add. The transform gets added to the list in the parent window i.e. 'Transformer Options' window.

Create Field	Allows creating a field under the user selected list type field. For example if a list called Football has two clubs namely, Knitters and Lambs, the user can add a new club called Irons and then the list has three clubs.
Sort	Performs sorting on values present in the selected field. In a complex list, the user needs to specify the key element for sorting while in case of simple list, the sorting takes place on the elements present in the list. The user can select the sort order as either ascending or descending. In the example of Football, when the list has three clubs, the user needs to select field 'name' under 'club' to sort the clubs based on name. The current club entries list as Irons, Knitters and Lambs if sort order is ascending and vice versa for sort order descending. Now, if the user wants the list of players sorted, the field 'player' needs to be selected and sort order defined for it
Sum	Performs summation of all the values present in the selected field. The output is stored in a field specified by the user. For example, if the user wants to view the total

points gained by each football club, user needs to select field 'points' under 'Tournament' and specify the output field name.

Copy　　　　Performs the copy operation from the selected field to the field specified by the user. When user selects a field to copy, the field and all fields under it (if any) are copied to the new field specified. This operation takes place at the same level of hierarchy.

Rename　　　Performs the rename operation of the selected field to the new name specified by the user.

The following XML code provides a reference to the List Transform feature:

```xml
<?xml version="1.0"?>
<sports_details>
  <sports name="football">
 <clubs>
   <club name="Knitters">
    <player>Samuel</player>
    <player>Messi</player>
    <player>kaka</player>
    <player>Alan</player>
    <coach>Stuart</coach>
    <Tournament name="Football League">
   <result>won</result>
   <points>4</points>
    </Tournament>
    <Tournament name="UEFA">
   <result>draw</result>
   <points>2</points>
    </Tournament>
   </club>
   <club name="Lambs">
    <player>Ronaldo</player>
    <player>Neymar</player>
    <player>Zlatan</player>
    <player>Mesut</player>
    <coach>Ivan</coach>
    <Tournament name="Airtel League">
   <result>draw</result>
   <points>2</points>
    </Tournament>
    <Tournament name="Champions League">
   <result>lost</result>
   <points>0</points>
    </Tournament>
   </club>
   <club name="Irons">
    <player>Scott</player>
    <player>Paul</player>
    <player>John</player>
    <player>Andrew</player>
    <coach>Jeff</coach>
```

```
  <Tournament name="CAF">
  <result>won</result>
  <points>4</points>
   </Tournament>
   <Tournament name="Copa America">
  <result>won</result>
  <points>4</points>
   </Tournament>
   </club>
</clubs>
 </sports>
 <sports name="badminton">
<clubs>
  <club name="Shuttlers">
   <player>Saina</player>
   <player>Viktor</player>
   <player>Chen</player>
   <player>Srikanth</player>
   <coach>Jan</coach>
   <Tournament name="Olympic Games">
  <result>won</result>
  <points>4</points>
   </Tournament>
   <Tournament name="Commonwealth Games">
  <result>won</result>
  <points>4</points>
   </Tournament>
   </club>
  <club name="Choppers">
   <player>Wang</player>
   <player>Sindhu</player>
   <player>Carolina</player>
   <player>Li Xuerui</player>
   <coach>Ratchanok</coach>
   <Tournament name="World Junior">
  <result>draw</result>
  <points>2</points>
   </Tournament>
   <Tournament name="Uber Cup">
  <result>draw</result>
  <points>2</points>
   </Tournament>
   </club>
  <club name="Lobbers">
   <player>Nozomi</player>
   <player>Chou</player>
   <player>Marc</player>
   <player>Lin</player>
   <coach>Kevin</coach>
   <Tournament name="World Senior">
  <result>won</result>
  <points>4</points>
   </Tournament>
```

```
    <Tournament name="Thomas Cup">
    <result>won</result>
    <points>4</points>
     </Tournament>
    </club>
 </clubs>
  </sports>
</sports_details>
```

Changing the Order of Transforms

If you have more than one transform to be executed on a particular output field, you can define the order in which they are executed.

> **Note:** If you map a single field to two different output fields (for example, ValidateAddress.City to Output.City1 and ValidateAddress.City to Output.City2), and you add transforms to each field, the transform for the secondary field must be executed first. You must change the execution order of the transforms to execute the second field transform (Output.City2) first.

1. Double-click the Transformer stage. The Transformer Options dialog box appears.
2. Select a transform and use the Move Up and Move Down buttons to rearrange the order of the transforms. The top transform will be executed first.

> **Note:** Dependent transforms cannot be moved above primary transforms (the transforms upon which the dependent transforms rely).

3. Click **OK**.

Creating a Custom Transform

The Transformer stage has predefined transforms that perform a variety of common data transformations. If the predefined transforms do not meet your needs, you can write a custom transform script using Groovy. This procedure describes how to create basic custom transforms using Groovy. For complete documentation on Groovy, see **groovy-lang.org**.

1. In Enterprise Designer, add a Transformer stage to the dataflow.
2. Double-click the Transformer stage.
3. Click **Add**.
4. Under **General**, click **Custom**.
5. In the **Custom transform name** field, enter a name for the transform you will create. The name must be unique.
6. Click **Script Editor**.

   This editor provides a variety of features to make developing your transform easier, such as code completion and palettes listing functions and fields.

Task	Instructions
To add a function	In the **Functions** pane, double-click the function you want to add.
	Note: The functions listed in the editor are functions provided to make writing custom transform scripts easier. They perform functions that would otherwise require multiple lines of Groovy code to accomplish. They are not standard Groovy functions.
To get the value from a dataflow field	In the **Input Fields** pane, double-click the input field you want. The following will be added to your script:
| | ```
data['FieldName']
``` |
| | For example, if you want to get the value from the field CurrentBalance, the following would be added: |
| | ```
data['CurrentBalance']
``` |
| **To set the value of a dataflow field** | Enter this code in the script editor: |
| | ```
data['FieldName']=NewValue
``` |
| | For example, to set the field Day to the day of the week contained in the field PurchaseDate: |
| | ```
data['Day']=dayOfWeek(data['PurchaseDate'])
``` |
	In this example, the function `dayOfWeek()` is used to get the day from the date value in the PurchaseDate field, and the result is written to the Day field.
	Tip: You can double-click the name of the output field in the **Output Fields** pane to add the field reference to the script.
To change the scope of a script variable in a dataflow	To change the scope of a script variable from a single input record to all the input records in a dataflow, use the *@Field* annotation in your script as shown:
| | ```
import groovy.transform.Field;
@Field ['data type']['VariableName']= Value;
``` |

Task	Instructions
	For example, to set the scope of a variable *RecordNumber* to a single input record, specify this:

```
int recordNumber = 1;
data['Record_Number']= recordNumber;
recordNumber++;
```

The output will be:

Date	Record_Number
3/14/18	1
3/15/18	1
3/16/18	1
3/17/18	1
3/18/18	1
3/19/18	1
3/20/18	1
3/21/18	1
3/22/18	1
3/23/18	1

To change the scope of this variable to all input records, specify this:

```
import groovy.transform.Field
@Field int recordNumber = 1;
data['Record_Number']= recordNumber;
recordNumber++;
```

The output will be:

Date	Record_Number
3/14/18	1
3/15/18	2
3/16/18	3
3/17/18	4
3/18/18	5
3/19/18	6
3/20/18	7
3/21/18	8
3/22/18	9
3/23/18	10

Task	Instructions
To create a new field using a numeric data type	Enter this code in the script editor:

```
data['FieldName'] = new constructor;
```

Where *constructor* is one of these:

java.lang.Double(*number*)

Task	Instructions

Creates a field with a data type of Double.

java.lang.Float(*number*)

Creates a field with a data type of Float.

java.lang.Integer(*number*)

Creates a field with a data type of Integer. You can also create a new integer field by specifying a whole number. For example, this will create an integer field with a value of 23:

```
data['MyNewField'] = 23;
```

java.lang.Long(*number*)

Creates a field with a data type of Long.

For example, to create a new field named "Transactions" with a data type of Double and the value 23.10, you would specify the following:

```
data['Transactions'] = new com.java.lang.Double(23.10);
```

To create a new field using a date or time data type

Enter this code in the script editor:

```
data['FieldName'] = new constructor;
```

Where *constructor* is one of these:

com.pb.spectrum.api.datetime.Date(*year*,*month*,*day*)

Creates a field with a data type of date. For example, December 23, 2013 would be:

```
2013,12,23
```

com.pb.spectrum.api.datetime.Time(*hour*,*minute*,*second*)

Creates a field with a data type of time. For example, 4:15 PM would be:

```
16,15,0
```

.

com.pb.spectrum.api.datetime.DateTime(*year*,*month*,*day*,*hour*,*minute*,*second*)

Task	Instructions

Creates a field with a data type of DateTime. For example, 4:15 PM on December 23, 2013 would be:

```
2013,12,23,16,15,0
```

For example, to create a new field named "TransactionDate" with a data type of Date and the value December 23, 2013, you would specify this:

```
data['TransactionDate'] = new
com.pb.spectrum.api.datetime.Date(2013,12,23);
```

To create a new field with a data type of Boolean

Enter this code in the script editor:

```
data['FieldName'] = true or false;
```

For example, to create a field named IsValidated and set it to false, you would specify this:

```
data['IsValidated'] = false;
```

To create a new list field

Use the `factory.create()` method to create new fields in a record then use the leftShift operator `<<` to append the new record to the list field.

```
NewListField = []

NewRecord = factory.create()
NewRecord['NewField1'] = "Value"
NewRecord['NewField12'] = "Value"
...
NewListField << NewRecord

NewRecord = factory.create()
NewRecord['NewField1'] = "Value"
NewRecord['NewField12'] = "Value"
...
NewListField << NewRecord
data['ListOfRecords'] = NewListField
```

For example, this creates a new list field called "addresses" consisting of two "address" records.

```
addresses = []
address = factory.create()
```

Task	Instructions

```
address['AddressLine1'] = "123 Main St"
address['PostalCode'] = "12345"
addresses << address

address = factory.create()
address['AddressLine1'] = "PO Box 350"
address['PostalCode'] = "02134"
addresses << address
data['Addresses'] = addresses
```

You can also create a new list field that contains a list of individual fields rather than a list of records. For example, this creates a new list field called PhoneNumbers containing home and work phone numbers:

```
phoneNumbers = []
phoneNumbers << data['HomePhone']
phoneNumbers << data['WorkPhone']
data['PhoneNumbers'] = phoneNumbers
```

To concatenate fields
Use the + symbol. For example, the following concatenates the FirstName field and the LastName field into a value and stores it in the FullName field

```
String fullname = data['FirstName'] + ' ' + data['LastName'];
data['FullName']=fullname;
```

In this example there are two input fields (AddressLine1 and AddressLine2) which are concatenated and written to the output field Address.

```
address1 = data['AddressLine1'];
address2 = data['AddressLine2'];
data['Address']=address1+ ',' + address2;
```

To parse a field
Identify a separation character then use `substring` to parse the field. In the following example, if the PostalCode field is greater than five characters, it separates the five-character ZIP Code and the +4 portion and writes them to separate fields in the output record.

```
if (data['PostalCode'].length() > 5)
{
  String postalCode = data['PostalCode'];
  int separatorPosition = postalCode.indexOf('-');
  String zip = postalCode.substring(0, separatorPosition);
  String plusFour = postalCode.substring(
```

Task	Instructions

```
      separatorPosition + 1,
      postalCode.length();
   data['Zip']=zip;
   data['PlusFour']=plusFour;
}
```

To perform conditional processing Use an `if` or `switch` statement. These are the most common conditional processing constructs. For more information see **groovy-lang.org**.

This example sets the field AddressCity to the first address line and city name if the city is Austin.

```
city = data['City'];
address1 = data['AddressLine1']
if(city.equals('Austin'))
data['AddressCity']=address1 +',' + city;
```

To perform looping Use the `for` loop. This is the only looping construct you should need. For more information about looping or syntax see **groovy-lang.org**.

To augment data Define a constant and use the concatenation character `+`. For example, the following script appends the word "Incorporated" to the FirmName field.

```
firmname = data['FirmName'];
constant = 'Incorporated';
if(firmname.length() > 0)
data['FirmName']=firmname + ' ' + constant;
```

To access an option specified at runtime If the dataflow has runtime options enabled, you can access settings passed to the dataflow at runtime by using this syntax:

```
options.get("optionName")
```

For example, to access an option named `casing`, you would include this in your custom transform script:

```
options.get("casing")
```

7. After you are done entering your script, click the "X" button in the window to close the editor.

8. In the **Input fields** field, select the field or fields to which you want to apply the transform.

9. In the **Output fields** field, specify the field to which you want to write the output from the transform. If necessary, you can define a new field by clicking the **Add** button to the right of the **Output fields** field.

10. When you are done, click the **Add** button at the bottom of the window.

11. Click **OK**.

Using a Mask Transform

You can use the Transformer stage to apply a mask transform to a field. A mask transform applies characters to a field, or removes characters from a field, using a specified pattern. For example, using a mask transform you could format a string of numbers like `8003685806` into a phone number like this: `(800) 368 5806`.

1. In Enterprise Designer, drag a Transformer stage to the canvas and connect it in the desired location.

2. Double-click the Transformer stage.

3. Click **Add**.

4. Expand **Formatting** and select **Mask**.

5. Select the type of mask you want to use.

Apply	Adds characters to a field to form the string into a new pattern.
Remove	Extracts a pattern of characters from a string.

6. In the **Mask string** field, specify the pattern you want to use when either adding characters or removing characters.

There are two types of characters you use when specifying the mask string: literal characters and mask characters.

Literal characters represent actual characters that are present in the string. When a remove mask is used, the input character must match the literal character exactly. If that is the case, then they will be removed from the input. Similarly, the literal characters will be added to the input in the position indicated by the mask definition when the apply mask is used.

The other type of character you can use in a mask string is a mask character. A mask character indicates the type of character that can be in a particular location of the input string. For example, if you have an input where the first character is a number, the first mask character needs to be `#`. Anything in the input that matches this mask character will be kept in the output.

The following table lists the mask characters you can use in the **Mask string** field:

Table 8: Mask Characters

Character	Definition
#	Any number.
'	Escape character, used to escape any of the special formatting characters.
U	Any character. All lowercase letters are mapped to upper case.
L	Any character. All upper case letters are mapped to lower case.
A	Any character or number.
?	Any character.
*	Anything.
H	Any hex character (0-9, a-f or A-F).

7. Click **Add**.
8. Click **OK**.

Mask Transform Examples

This is an apply mask that applies formatting to a string. Because "(" and ")" and <space> are literals, they will be added to the output. All the numbers will be kept because # is a mask character.

Input: `8003685806`
Mask string: `(###) ### ####`
Output: `(800) 368 5806`

The following example is a remove mask that removes the dash in the ZIP Code.

Input: `60510-1135`
Mask string: `*****-****`
Output: `605101135`

Unique ID Generator

The Unique ID Generator stage creates a unique key that identifies a specific record. A unique ID is crucial for data warehouse initiatives in which transactions may not carry all name and address data, but must be attributed to the same record/contact. A unique ID may be implemented at the individual, household, business, and/or premises level. Unique ID Generator provides a variety of algorithms to create unique IDs.

The unique ID is based on either a sequential number or date and time stamp. In addition, you can optionally use a variety of algorithms to generate data to appended to the ID, thereby increasing the likelihood that the ID will be unique. The sequential number or date and time stamp IDs are required and cannot be removed from the generated ID.

Unique ID Generator can be used to generate a non-unique key using one of the key generation algorithms. In non-unique mode, you can create keys to use for matching. This may be useful in a data warehouse where you have already added keys to a dimension and you want to generate a key for new records in order to see if the new records match an existing record.

The following example shows that each record in the input is assigned a sequential record ID in the output.

Record	RecordID
John Smith	0
Mary Smith	1
Jane Doe	2
John Doe	3

The Unique ID stage produces a field named RecordID which contains the unique ID. You can rename the RecordID field as required.

Defining a Unique ID

By default, the Unique ID Generator stage creates a sequential ID, with the first record having an ID of 0, the second record having an ID of 1, the third record having an ID of 2, and so on. If you want to change how the unique ID is generated, follow this procedure.

1. In the Unique ID Generator stage, on the **Rules** tab, click **Modify**.
2. Choose the method you want to use to generate the unique ID.

Options	Description
Sequential Numeric tag starting at	Assigns an incremental numeric value to each record starting with the number you specify. If you specify 0, the first record will have an ID of 0, the second record will have an ID of 1, and so on.

Options	Description
Sequential Numeric tag starting at value in a database field	

Options	Description
	Assigns an incremental numerical value to each record starting with the maximum number read from the database field. This number is then incremented by 1 and assigned to the first record. For example, if the number read from the database field is 30, the first record will have an ID of 31, the second record will have an ID of 32, and so on.
Connection	Select the database connection you want to use. Your choices vary depending on what connections are defined in the Connection Manager of Management Console. If you need to make a new database, or modify or delete an existing connection, click **Manage**.

If you are adding or modifying a database connection, complete these fields:

Connection name

Enter a name for the connection. This can be anything you choose.

Database driver

Select the appropriate database type.

Connection options

Specify the host, port, instance, user name, and password to use to connect to the database.

Options	Description
Table view	Specifies the table or view in the database that you want to query.
Database field	Select a column from the list to generate a unique key.

The supported datatypes for unique ID generation are:

long	A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and $2^{63}-1$ (9,223,372,036,854,775,807).
integer	A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and $2^{31}-1$ (2,147,483,647).
bigdecimal	A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.
double	A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52}) \times 2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.
float	A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23}) \times 2^{127}$. In E notation, the range

Options	Description
	of values -3.402823E+38 to 3.402823E+38.
Date/Time stamp	Creates a unique key based on the date and time stamp instead of sequential numbering.
UUID	Creates a universally unique 32-digit identifier key for each record. The digits in the key are displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and four hyphens). Example: 123e4567-e89b-12d3-a456-432255330000
Off	Select this option only if you want to generate a non-unique key using an algorithm.

3. Click **OK**.

Using Algorithms to Augment a Unique ID

Unique ID Generator generates a unique ID for each record by either numbering each record sequentially or generating a date/time stamp for each record. You can optionally use algorithms to append additional information to the sequential or date/time unique ID, thereby creating a more complex unique ID and one that is more likely to be truly unique.

1. In the Unique ID Generator stage, click **Add**.
2. In the **Algorithm** field, select the algorithm you want to use to generate additional information in the ID. One of the following:

Consonant	Returns specified fields with consonants removed.
Double Metaphone	Returns a code based on a phonetic representation of their characters. Double Metaphone is an improved version of the Metaphone algorithm, and attempts to account for the many irregularities found in different languages.
Koeln	Indexes names by sound as they are pronounced in German. Allows names with the same pronunciation to be encoded to the same representation so that they can be matched, despite minor differences in spelling. The result is always a sequence of numbers; special characters and white spaces are ignored. This option was developed to respond to limitations of Soundex.
MD5	A message digest algorithm that produces a 128-bit hash value. This algorithm is commonly used to check data integrity.
Metaphone	Returns a Metaphone coded key of selected fields. Metaphone is an algorithm for coding words using their English pronunciation.

Metaphone (Spanish)	Returns a Metaphone coded key of selected fields for the Spanish language. This metaphone algorithm codes words using their Spanish pronunciation.
Metaphone 3	Improves upon the Metaphone and Double Metaphone algorithms with more exact consonant and internal vowel settings that allow you to produce words or names more or less closely matched to search terms on a phonetic basis. Metaphone 3 increases the accuracy of phonetic encoding to 98%. This option was developed to respond to limitations of Soundex.
Nysiis	Phonetic code algorithm that matches an approximate pronunciation to an exact spelling and indexes words that are pronounced similarly. Part of the New York State Identification and Intelligence System. Say, for example, that you are looking for someone's information in a database of people. You believe that the person's name sounds like "John Smith", but it is in fact spelled "Jon Smyth". If you conducted a search looking for an exact match for "John Smith" no results would be returned. However, if you index the database using the NYSIIS algorithm and search using the NYSIIS algorithm again, the correct match will be returned because both "John Smith" and "Jon Smyth" are indexed as "JAN SNATH" by the algorithm.
Phonix	Preprocesses name strings by applying more than 100 transformation rules to single characters or to sequences of several characters. 19 of those rules are applied only if the character(s) are at the beginning of the string, while 12 of the rules are applied only if they are at the middle of the string, and 28 of the rules are applied only if they are at the end of the string. The transformed name string is encoded into a code that is comprised by a starting letter followed by three digits (removing zeros and duplicate numbers). This option was developed to respond to limitations of Soundex; it is more complex and therefore slower than Soundex.
Soundex	Returns a Soundex code of selected fields. Soundex produces a fixed-length code based on the English pronunciation of a word.
Substring	Returns a specified portion of the selected field.

3. In the **Field name** field, choose the field to which you want to apply the algorithm. For example, if you chose the soundex algorithm and chose a field named City, the ID would be generated by applying the soundex algorithm to the data in the City field.

4. If you selected the substring algorithm, specify the portion of the field you want to use in the substring:

   a) In the **Start position** field, specify the position in the field where you want the substring to begin.

   b) In the **Length** field, select the number of characters from the start position that you want to include in the substring.

   For example, say you have the following data in a field named LastName:

   ```
   Augustine
   ```

If you specified 3 as the start position and 6 as the end position, the substring would produce:

```
gustin
```

5. Check the **Remove noise characters** box to remove all non-numeric and non-alpha characters such as hyphens, white space, and other special characters from the field before applying the algorithm.

6. For consonant and substring algorithms, you can sort the data in the field before applying the algorithm by checking the **Sort input** box. You can then choose to sort either the characters in the field or terms in the field in alphabetical order.

7. Click **OK** to save your settings.

8. Repeat as needed if you want to add additional algorithms to produce a more complex ID.

> **Note:** The unique key definition is always displayed in a different color and cannot be deleted.

Defining a Non-Unique ID

Unique ID Generator can be used to generate a non-unique key using one of the key generation algorithms. In non-unique mode, you can create keys to use for matching. This may be useful in a data warehouse where you have already added keys to a dimension and you want to generate a key for new records in order to see if the new records match an existing record.

1. In the Unique ID Generator stage, on the **Rules** tab, click **Modify**.

2. Select **Off**.

   This turns off the unique ID portion of the ID generation rules. With this option off, only the algorithm you choose in the following steps will be used to create the ID. This means that any records that have the same data in the fields you use to generate the ID will have the same ID. You can then use the ID for matching.

3. Click **OK**.

4. At the warning prompt, click **Yes**.

5. In the Unique ID Generator stage, click **Add**.

6. In the **Algorithm** field, select the algorithm you want to use to generate additional information in the ID. One of the following:

Consonant	Returns specified fields with consonants removed.
Double Metaphone	Returns a code based on a phonetic representation of their characters. Double Metaphone is an improved version of the Metaphone algorithm, and attempts to account for the many irregularities found in different languages.
Koeln	Indexes names by sound as they are pronounced in German. Allows names with the same pronunciation to be encoded to the same representation so that they can be matched, despite minor differences in spelling. The result is always

	a sequence of numbers; special characters and white spaces are ignored. This option was developed to respond to limitations of Soundex.
MD5	A message digest algorithm that produces a 128-bit hash value. This algorithm is commonly used to check data integrity.
Metaphone	Returns a Metaphone coded key of selected fields. Metaphone is an algorithm for coding words using their English pronunciation.
Metaphone (Spanish)	Returns a Metaphone coded key of selected fields for the Spanish language. This metaphone algorithm codes words using their Spanish pronunciation.
Metaphone 3	Improves upon the Metaphone and Double Metaphone algorithms with more exact consonant and internal vowel settings that allow you to produce words or names more or less closely matched to search terms on a phonetic basis. Metaphone 3 increases the accuracy of phonetic encoding to 98%. This option was developed to respond to limitations of Soundex.
Nysiis	Phonetic code algorithm that matches an approximate pronunciation to an exact spelling and indexes words that are pronounced similarly. Part of the New York State Identification and Intelligence System. Say, for example, that you are looking for someone's information in a database of people. You believe that the person's name sounds like "John Smith", but it is in fact spelled "Jon Smyth". If you conducted a search looking for an exact match for "John Smith" no results would be returned. However, if you index the database using the NYSIIS algorithm and search using the NYSIIS algorithm again, the correct match will be returned because both "John Smith" and "Jon Smyth" are indexed as "JAN SNATH" by the algorithm.
Phonix	Preprocesses name strings by applying more than 100 transformation rules to single characters or to sequences of several characters. 19 of those rules are applied only if the character(s) are at the beginning of the string, while 12 of the rules are applied only if they are at the middle of the string, and 28 of the rules are applied only if they are at the end of the string. The transformed name string is encoded into a code that is comprised by a starting letter followed by three digits (removing zeros and duplicate numbers). This option was developed to respond to limitations of Soundex; it is more complex and therefore slower than Soundex.
Soundex	Returns a Soundex code of selected fields. Soundex produces a fixed-length code based on the English pronunciation of a word.
Substring	Returns a specified portion of the selected field.

7. In the **Field name** field, choose the field to which you want to apply the algorithm. For example, if you chose the soundex algorithm and chose a field named City, the ID would be generated by applying the soundex algorithm to the data in the City field.

8. If you selected the substring algorithm, specify the portion of the field you want to use in the substring:

a) In the **Start position** field, specify the position in the field where you want the substring to begin.

b) In the **Length** field, select the number of characters from the start position that you want to include in the substring.

   For example, say you have the following data in a field named LastName:

   `Augustine`

   If you specified 3 as the start position and 6 as the end position, the substring would produce:

   `gustin`

9. Check the **Remove noise characters** box to remove all non-numeric and non-alpha characters such as hyphens, white space, and other special characters from the field before applying the algorithm.

10. For consonant and substring algorithms, you can sort the data in the field before applying the algorithm by checking the **Sort input** box. You can then choose to sort either the characters in the field or terms in the field in alphabetical order.

11. Click **OK** to save your settings.

12. Repeat as needed if you want to add additional algorithms to produce a more complex ID.

> **Note:** The unique key definition is always displayed in a different color and cannot be deleted.

Flow Output

To define the output from a dataflow, use a "sink" stage. A sink is the last stage in a dataflow. It defines what to do with the output from the dataflow. A sink can also perform other actions at the end or a dataflow, such as executing a program.

Output from a Job

Output from a job can be written to a file or a database. Spectrum™ Technology Platform has the ability to write data to many file formats and database types. The types of sinks you can write to depend on which modules you have licensed. See the solution guide for your modules available at **support.pb.com/spectrum**.

Output from a Service

Output data from a service is defined in an Output stage. This stage defines the fields that the service will return in response to a web service request or an API call.

Defining Service Output

The Output stage defines the output fields that the service or subflow returns. Follow the steps below to define the service output.

1. Double-click the Output icon on the canvas. The **Output Options** dialog box appears. When you open the **Output Options** dialog box for the first time, a list of fields defined in the Input is displayed.
2. To add a new field to the field list, click **Add**. The **Add Custom Field** dialog box appears. You can also modify or delete a custom field.
3. Click **Add** again.
4. Type the field name in the text box.
5. Select the **Data type** and press **OK**. The following data types are supported:

bigdecimal	A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.
boolean	A logical type with two values: true and false.
bytearray	An array (list) of bytes.

> **Note:** Bytearray is not supported as an input for a REST service.

date	A data type that contains a month, day, and year. For example, 2012-01-30 or January 30, 2012. You can specify a default date format in Management Console.
datetime	A data type that contains a month, day, year, and hours, minutes, and seconds. For example, 2012/01/30 6:15:00 PM.
double	A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52})\times2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.
float	A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23})\times2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.
integer	A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647).
list	Strictly speaking, a list is not a data type. However, when a field contains hierarchical data, it is treated as a "list" field. In Spectrum™ Technology Platform a list is a collection of data consisting of multiple values. For example, a field

Names may contain a list of name values. This may be represented in an XML structure as:

```
<Names>
     <Name>John Smith</Name>
     <Name>Ann Fowler</Name>
</Names>
```

It is important to note that the Spectrum™ Technology Platform list data type different from the XML schema list data type in that the XML list data type is a simple data type consisting of multiple values, whereas the Spectrum™ Technology Platform list data type is similar to an XML complex data type.

long A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807).

string A sequence of characters.

time A data type that contains the time of day. For example, 21:15:59 or 9:15:59 PM.

You can also add a new, user-defined data type if necessary, and that new type can be a list of any defined data type. For example, you could define a list of names (string), or a new data type of addresses that includes AddressLine1 (string), City (string), StateProvince (string) and PostalCode (string). After you create the field, you can view the data type by accessing the Input Options dialog and pressing the button in the Data Type column. The **Data Type Details** dialog box will appear, showing the structure of the field.

6. Click **OK** again.
7. Click the check box next to **Expose** to select the check box of all fields in the field list. Selecting a field in the field list exposes it to the dataflow for stage operations. Click the check box again to clear the check box for all fields in the list. Clearing the check box of one or more fields in the field list and clicking **OK** deletes the field from the field list.

> **Note:** If you define hierarchical data in the input fields, you will not be able to import data or view the data vertically.

8. Click **OK** to return to the canvas.

Defining A Web Service Data Type

The **Data type name** field allows you to control the WSDL (SOAP) and WADL (REST) interfaces for the service you are creating. The name of the Rows element is determined by the name you give this stage in the service, and the name of the Row element is determined by the text you enter here.

> **Note:** For WSDL, both requests and responses are affected, but for WADL only responses are affected.

Prior to naming this stage and entering text in this field, your code might look like this:

```
<Rows>
    <Row>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
    </Row>
    <Row>
        <FirstName>Jane</FirstName>
  <LastName>Doe></LastName>
    </Row>
</Rows>
```

After naming this stage and entering text in this field, your code might look like this:

```
<Names>
    <Name>
        <FirstName>John</FirstName>
        <LastName>Doe</LastName>
    </Name>
    <Name>
        <FirstName>Jane</FirstName>
        <LastName>Doe></LastName>
    </Name>
</Names>
```

Running an External Program

An Execute Program stage invokes an executable, such as a program or command line command, when it receives a record. To use an Execute Program stage in your dataflow:

Options

Option	Description
Command-line	The executable name and arguments (if applicable). The arguments can be data available in the dataflow. To access that data, click the [...] (Browse) button. You can select from the following three contexts: Current Job ID, Current Job Name, or Current User Name. You can also select from the available fields. For example, JobStatus and JobComment.

Option	Description
Timeout	Specifies whether to cancel the execution if the command does not respond within a given amount of time. One of the following:

	No timeout	Do not cancel the execution if the command fails to respond.
	Timeout in milliseconds	Cancels the execution attempt if the command does not respond in the specified number of milliseconds.

Option	Description
Environment Variables	Optional. Specifies environment variables to use when executing the command. To add an environment variable click **Add**. Enter the appropriate key word in the **Key** field. An example might be "JAVA_HOME". Enter the appropriate value in the **Value** field. An example might be `C:\Java\jre7`. Alternatively, you can select a field from the Field List dialog box by clicking the [...] (Browse) button. You can select from the following three contexts: Current Job ID, Current Job Name, or Current User Name. You can also select from the available fields. For example, JobStatus and JobComment.

Terminating a Job Based on a Condition

The Terminate Job stage is used in combination with Conditional Router to end a job if certain criteria are found within a record. If a record is sent to Terminate Job, the job ends.

> **Note:** Terminate Job is not available in services or subflows.

To use Terminate Job, add a Conditional Router and a Terminate Job stage to your dataflow. Then connect the stages and configure the Conditional Router. The Conditional Router should be configured to contain the criteria you want to trigger job termination. When a record is found that meets the criteria, it is passed to Terminate Job and the job terminates, producing a message that says "Job terminated by stage: <stage label>." The completed dataflow should look something like this:

Discarding Records

The Write to Null stage discards records. Records are counted but discarded. Use this stage if there are records that you do not want to preserve after the dataflow finishes.

Embedded Dataflows

An embedded dataflow reduces the number of stages displayed on the Enterprise Designer canvas at one time by grouping stages together. The grouped stages then appear as a single stage. You can use embedded dataflows to:

• Simplify the layout of complex dataflows by grouping stages together and having them represented as one stage on the canvas.
• Process records in groups using the iteration feature.
• Use a value in a field to set stage options in the embedded dataflow.

You can add an unlimited number of embedded dataflows to a dataflow, and you can put embedded dataflows within embedded dataflows.

Differences Between Embedded Dataflows and Subflows

There are two major differences between an embedded dataflow and a subflow. First, iteration processing is only available in embedded dataflows. Iteration allows you to process groups of records together for purposes such as aggregating records for processing or setting stage options based on field values. The second difference between an embedded dataflow and subflow is that an embedded dataflow cannot be used in more than one dataflow. If you want to reuse a portion of a dataflow in multiple dataflows, create a subflow instead of an embedded dataflow. Embedded dataflows can be converted into subflows if you decide you want to reuse an embedded dataflow in other dataflows, but when you convert an embedded dataflow to a subflow its iterations options are removed.

Grouping Stages into an Embedded Dataflow

An embedded dataflow groups stages together into a single stage, allowing you to simplify the layout of complex dataflows and set processing options using field values.

1. In your dataflow, add the stages that you want to convert to an embedded dataflow.
2. Select the stages you want to convert to an embedded dataflow by clicking and dragging a box around the stages.

For example, this would select the Sorter and Transformer stages to convert to an embedded dataflow.

Note: Report stages cannot be included in an embedded dataflow.

3. Right-click one of the selected stages and select **Group into Embedded Dataflow**
4. Enter a name for the embedded dataflow. The name will be used as the label for the embedded dataflow stage on the canvas.
5. Click **OK**.

The stages you selected are now grouped into an embedded dataflow. The following example shows an embedded dataflow named SortTansform.

Editing an Embedded Dataflow

An embedded dataflow groups stages together into a single stage, allowing you to simplify the layout of complex dataflows and set processing options using field values.

1. Right-click the icon and select **Edit This Embedded Dataflow**.

   Tip: Alternatively, you can use the breadcrumb links at the top of the dataflow to open embedded dataflows. For example, this shows how to open an embedded dataflow named SortTransform:

When you open an embedded dataflow you will see an Input stage and an Output stage. These represent the input to the embedded dataflow from the parent dataflow, and the output to the parent dataflow from the embedded dataflow.

2. Modify the embedded dataflow as needed.

Any changes you make to an embedded dataflow are saved the next time you save the parent dataflow.

Using Iteration with an Embedded Dataflow

Iteration settings specify how an embedded dataflow should process incoming records. By default, an embedded dataflow processes each record individually just as any other stage in the dataflow would. But if you use iteration, you can process groups of records together, which can be useful for things like performing comparisons or calculations based on groups of records rather then the entire set of input data. You can also use iteration to set stage options based on the data in each record.

There are two kinds of iteration: per-record iteration and per-group iteration. In per-record iteration, an embedded dataflow process one record at a time and the result is sent along to the next stage following the embedded dataflow. Per-record iteration is useful if you want to set stage options on a record-by-record basis using field values.

In per-group iteration, records are grouped by a key field and the embedded dataflow processes each group. All the records in a group are processed in one iteration, then the group is written to the next stage following the embedded subflow. Use per-group iteration to perform processing on groups of related records, as well as to set stage options to use when processing the group of records. For example, you might want to group records by customer ID so that you can perform an analysis of each customer's records, perhaps to determine which store each customer visits most often.

You should consider the impact on performance when using iteration. Each time a new iteration starts, there is some overhead during the initialization of the embedded dataflow, and this overhead

can be significant, especially if you have embedded dataflows within other embedded dataflows. For example, if the an embedded dataflow iterates 1,000 times and it contains within it another embedded dataflow that also iterates 1,000 times, the total number of iterations would be 1,000,000. Using per-record iteration has a more significant impact on performance since each record kicks off a new iteration.

1. Create an embedded dataflow containing the stage or stages that you use for iteration.

   > **Note:** There are some limitations to what can be included in embedded dataflows that have iteration enabled:
   >
   > • The Stream Combiner stage cannot be the first stage in an embedded dataflow that has iteration enabled.
   > • The embedded dataflow cannot contain a sink that writes to a file located on the client. Sinks inside an embedded dataflow must write to a file on the Spectrum™ Technology Platform server or on a file server.

2. Double-click the embedded dataflow icon.
3. Check the **Enable iteration** check box.
4. If there is more than one input channel connected to the embedded dataflow, use the **Port** field to choose the port whose records you want to use to drive iteration.

   For example, say you have two input ports, A and B, and you choose to iterate each time a key field changes. If you choose to use port B for iteration, the embedded dataflow will start a new iteration each time a key field in the records from port B changes. All the records from the other port, port A, will be read into the embedded dataflow, cached, and used for each iteration.

5. Select the type of iteration you want to perform.

Iterate each time a key field changes	In this type of iteration, the embedded dataflow processes groups of records that have the same value in one or more fields. When the embedded dataflow finishes processing the group of records, the embedded dataflow resets and a new group of records is processed. Use this type of iteration to create embedded dataflows that process groups of records and then outputs each record group separately.
	Tip: If you choose this type of iteration, you can improve performance by placing a Sorter stage in front of the embedded dataflow and sorting the records by the key field.
Iterate per record	In this type of iteration, the embedded dataflow processes one record at a time. Every time one record completes the embedded dataflow processing, the result is sent to the output and a new record is processed. Embedded dataflows that iterate per record handle each record as a new dataflow execution.

6. If you choose **Iterate each time a key field changes**, check the box **Ignore case when comparing values** if you want to ignore differences in case when evaluating key field values to determine record groups.

7. Specify one or more key fields.

    a) Click **Add**.

    b) Choose the field you want to use as a key field.

    c) If you want to use the field's value to set a stage option within the embedded dataflow, specify the name of the option you want to set.

    d) Click **OK**.

    e) Add additional key fields if needed.

       If you have more than one key field and you chose the option **Iterate each time a key field changes**, records must contain the same value in all key fields to be grouped together.

Ungrouping an Embedded Dataflow

When you ungroup an embedded dataflow, the stages from the embedded dataflow are placed in the parent dataflow at the location of the embedded dataflow. Any iteration settings you may have configured for the embedded dataflow are removed.

To ungroup the stages from an embedded dataflow, right-click the embedded dataflow and select **Ungroup this Embedded Dataflow**.

Converting an Embedded Dataflow to a Subflow

If you want to reuse an embedded dataflow in another dataflow, you must convert the embedded dataflow to a subflow. This is because embedded dataflows cannot be used in other dataflows. Once you convert an embedded dataflow to a subflow, it can be used like any other subflow.

> **Note:** If the embedded dataflow has iteration enabled, iteration settings will be removed when it is converted to a subflow. Subflows do not support iteration.

1. In Enterprise Designer, open the dataflow containing the embedded dataflow that you want to convert to a subflow.

2. Right-click the embedded dataflow and select **Convert Stage to Subflow**.

3. Enter a name for the new subflow and click **OK**.

The embedded dataflow is converted to a subflow and is made available in the Enterprise Designer palette under the User-Defined Stages folder.

Reports

Spectrum™ Technology Platform provides reporting capabilities for jobs. You can use standard reports that come with some modules or you can design your own reports. When a report is included in a dataflow the entire dataflow runs, and after completion the report stages in the dataflow are executed and the reports are saved in the format you choose, for example PDF.

Adding a Standard Report to a Job

A standard report is a pre-configured report that is included with a Spectrum™ Technology Platform module. For example, the Location Intelligence Module includes the Point In Polygon Summary report, which summarizes the results of point in polygon calculations, such as the number of polygon matches, the database used for the job, and other information.

The following procedure describes how to add a standard report to a job.

1. In Enterprise Designer, on the left side of the window under Palette, click **Reports**.
   A list of available reports appears.
2. Drag the report you want onto the canvas. You do not need to connect the report icon to anything.
3. Double-click the report.
4. Select the stages that you want to contribute to the report.
5. Click the **Parameters** tab.
6. Clear the **Use default reporting options** check box and select the appropriate output format if you wish to specify a format other than PDF (such as html or txt).

Setting Report Options for a Job

Reports provide summary information about a job, such as the number of records processed, the settings used for the job, and so on. Report options specify how to handle the reports generated by a job, such as the output format and archiving options. Default values for report options are specified in Management Console but you can override the default options for a job in Enterprise Designer.

The following procedure describes how to specify report options for a job.

1. Open the job in Enterprise Designer and go to **Edit** > **Job Options**.
2. Click the **Reporting** tab.
3. Clear the **Use global reporting options** check box.

4. Choose the format you want to use to save reports. Reports can be saved as HTML, PDF, or text.

5. Choose where you want to save reports.

Save reports to job history	Saves reports on the server as part of the job history. This makes it convenient for Management Console and Enterprise Designer users to view reports since the reports are available in the execution history.
Save reports to a file	Saves reports to a file in the location you specify. This is useful if you want to share reports with people who are not Spectrum™ Technology Platform users. It is also useful if you want to create an archive of reports in a different location. To view reports saved in this manner you can use any tool that can open the report's format, such as a PDF viewer for PDF reports or a web browser for HTML reports.

6. If you selected **Save reports to a file**, complete these fields.

Report location	The folder where you want to save reports.
Append to report name	Specifies variable information to include in the file name. You can choose one or more of the following:

Job ID	A unique ID assigned to a job execution. The first time you run a job on your system the job has an ID of 1. The second time you run a job, either the same job or another job, it has a job ID of 2, and so on.
Stage	The name of the stage that contributed data to the report, as specified in the report stage in Enterprise Designer.
Date	The day, month, and year that the report was created.

Overwrite existing reports	Replaces previous reports that have the same file name with the new report. If you do not select this option and there is an existing report that has the same name as the new report, the job will complete successfully but the new report will not be saved. A comment will appear in the execution history indicating that the report was not saved.

7. Click **OK**.

When you run your job, the Execution History will contain a column that shows if there are any reports that are associated with the job. An empty icon indicates no reports, one document icon indicates one report, and multiple documents icons indicate multiple reports. You can use the Job Detail to view, save, or print the report.

> **Note:** To delete a report, right-click the report icon on the canvas and select **Delete**.

Viewing Reports

To view reports, first run the job then do one of the following:

- In Enterprise Designer, the **Execution Details** window will appear when you run your job. Select the report you want to view.
- In the Management Console, in the Execution node, click **History** then select the job whose reports you want to view, then click **Details**.

Using Custom Reports

Spectrum™ Technology Platform modules come with reports that are useful for basic reporting. However, if you have report requirements that are not met by the standard reports, you can create your own custom reports and include them in your dataflow.

1. Create the report template using the report design tool of your choice. Your design tool must be able to export the report in the JasperReports format (.jrxml).
2. Copy your .jrxml file to the `server\app\import` folder on the Spectrum™ Technology Platform server.

   Within a few seconds, the report template will be imported into the system and made available in Enterprise Designer.

3. In Enterprise Designer, open the job to which you want to add your custom report.
4. On the left side of the window, under Palette, click **Reports**.
5. Drag your custom report to the canvas.
6. Specify the data source for the report by doing one of the following:

Option	Description
To report on the dataflow's input	Connect the report to the source stage you want to report on using the gray diamond-shaped report port as shown here: The report will be based on the dataflow's input data and will not reflect any of the processing that occurs in the dataflow.

Option	Description
To report on the dataflow's output	Connect the report to the sink stage you want to report on using the gray diamond-shaped report port as shown here: The report will be based on the dataflow's output data and will reflect the dataflow's effect on the data.
To use a query embedded in the report template	If the report template contains an embedded SQL query in the `<queryString>` element of the JRXML file, double-click the report icon and check the **Use embedded query** box, then select the database connection to use for the query. **Note:** If you need to define a database connection, open the Management Console and go to **Resources**, then **Connections**.

You can connect multiple reports to a source or sink, as shown here:

7. If the report contains user-defined parameters:
   a) Double-click the report icon on the canvas.
   b) On the **Parameters** tab, specify the values you want to use for the report's user-defined parameters.

8. Optional: If necessary, right-click on the channel and map the fields from the source or sink to the fields in the report.

Performance Considerations

Design Guidelines for Optimal Performance

Carefully designing your dataflows to optimize performance is the most important thing you can do to achieve good performance on Spectrum™ Technology Platform. These guidelines describe techniques you can use optimize dataflow performance.

Minimize the Number of Stages

Spectrum™ Technology Platform achieves high performance through parallel processing. Each stage in a flow runs asynchronously in its own thread. However, it is possible to overthread the processors when executing certain types of dataflows. When this happens, the system spends as much or more time managing threads as doing "real work". We have seen dataflows with as many as 130 individual stages that perform very poorly on smaller servers with one or two processors.

So the first consideration in designing dataflows that perform well is to use as many stages as needed, but no more. Some examples of using more stages than needed are:

• Using multiple conditional routers where one would suffice
• Defining multiple transformer stages instead of combining the transforms in a single stage

Fortunately it is usually possible to redesign these dataflows to remove redundant or unneeded stages and improve performance.

For complex flows, consider using embedded flows or subflows to reduce clutter on the canvas and make it easier to view and navigate the flow. Using embedded flows does not have a performance benefit at runtime, but it does make it easier to work with flows in Enterprise Designer. Using subflows to simplify complex flows can improve Enterprise Designer performance when editing flows.

Reduce Record Length

Since data is being passed between concurrently executing stages, another consideration is the length of the input records. Generally input with a longer record length will take longer to process than input with a shorter record length, simply because there is more data to read, write, and sort. Dataflows with multiple sort operations will particularly benefit from a reduced record length. In the case of very large record lengths it can be faster to remove the unnecessary fields from the input prior to running the Spectrum™ Technology Platform job then append them back to the resulting output file.

Use Sorting Appropriately

Another consideration is to minimize sort operations. Sorting is often more time consuming than other operations, and can become problematic as the number and size of input records increases. However, many Spectrum™ Technology Platform stages either require or prefer sorted input data. The Universal Addressing Module and Enterprise Geocoding Module, for example, perform optimally when the input is sorted by country and postal code. Stages such as Intraflow Match and Interflow Match require that the input be sorted by the "group by" field. In some cases you can use an external sort application to presort the input data and this can be faster than sorting within the Spectrum™ Technology Platform dataflow.

Stage Runtime Performance Options

Runtime performance options control how individual stages in a dataflow are executed and provide settings you can use to improve the performance of your dataflow. The settings available to you depend on how your Spectrum™ Technology Platform environment has been configured.

- The **Local** option is the default setting in which stages run on the local Spectrum™ Technology Platform server and use one runtime instance. The runtime instances setting can be increased, thereby utilizing parallel processing and improving performance.
- The **Distributed** option is typically used in a clustered environment which involves installing a load balancer and multiple Spectrum™ Technology Platform servers.
- The **Remote** option can be used if your environment consists of multiple Spectrum™ Technology Platform servers but is not configured for distributed processing. This option allows you to have a stage's processing performed by another server.

Database Pool Size and Runtime Instances

In most Spectrum™ Technology Platform environments there are multiple flows running at the same time, whether they are batch jobs or services responding to web service or API requests. To optimize concurrent processing, you can use the database pool size setting, which limits the number of concurrent requests a Spectrum database handles, and runtime instances, which controls the number of instances of a flow stage that run concurrently. These two settings should be tuned together to achieve optimal performance.

Database Pool Size

Spectrum databases contain reference data used by certain stages, such as postal data used to validate addresses, or geocoding data used to geocode addresses. These databases can be configured to accept multiple concurrent requests from the dataflow stages or services that use them, thereby improving the performance of the dataflows or service requests. The database pool size sets the maximum number of concurrent requests that a Spectrum database will process. By default, Spectrum databases have a pool size of 4, meaning the database can process four requests simultaneously.

The optimal pool size varies by module. You will generally see the best results by setting the pool size between one-half to twice the number of CPUs on the server, with the optimal pool size for most modules being the same as the number of CPUs. For example, if your server has four CPUs you may want to experiment with a pool size between 2 (one-half the number of CPUs) and 8 (twice the number of CPUs) with the optimal size possibly being 4 (the number of CPUs).

When modifying the pool size you must also consider the number of runtime instances specified in the dataflow for the stages accessing the database. Consider for example a dataflow that has a Geocode US Address stage that is configured to use one runtime instance. If you set the pool size for the US geocoding database to four, you will not see a performance improvement because there would be only one runtime instance and therefore there would only be one request at a time to the database. However, if you were to increase the number of runtime instances of Geocode US Address to four, you might then see an improvement in performance since there would be four instances of Geocode US Address accessing the database simultaneously, therefore using the full pool.

Runtime Instances

Each stage in a dataflow operates asynchronously in its own thread and is independent of any other stage. This provides for parallel processing of stages in a dataflow, allowing you to utilize more than one runtime instance for a stage. This is useful in dataflows where some stages process data faster than others. This can lead to an unbalanced distribution of work among the threads. For example, consider a dataflow consisting of the following stages:

Read from File   Validate Address   Geocode US Address   Write to File

Depending on the configuration of the stages, it may be that the Validate Address stage processes records faster than the Geocode US Address stage. If this is the case, at some point during the execution of the dataflow all the records will have been processed by Validate Address, but Geocode US Address will still have records to process. In order to improve performance of this dataflow, it is necessary to improve the performance of the slowest stage - in this case Geocode US Address. One way to do that is to specify multiple runtime instances of the stage. Setting the number of runtime instances to two, for example, means that there will be two instances of that stage, each running in its own thread, available to process records.

As a general rule, the number of runtime instances should be at least equal to the number of instances of the remote component. See the *Administration Guide* for information about remote components. While specifying multiple runtime instances can help improve performance, setting this value too high can strain your system resources, resulting in decreased performance.

> **Note:** Using multiple runtime instances only improves performance when running jobs or when running service requests with more than one record.

Tuning Procedure

Finding the right settings for database pool size and runtime instances is a matter of experimenting with different settings to find the ones maximize available server resources without overloading resources and causing reduced performance.

> **Note:** You should optimize the dataflow pool size before tuning the database pool size. For information about optimizing the dataflow pool size, see **Dataflow Pool Size**.

1. Begin by finding sample data to use as you test different settings. The sample dataset should be large enough that execution time is measurable and can be validated for consistency. The sample data should also be representative of the actual data you want to process. For example, if you are doing performance testing for geocoding, be sure that your test data has an equal number of records for all the countries you intend to geocode.
2. If you are testing a service or dataflow that requires the use of a database resource, such as postal databases or geocoding databases, make sure that you have the latest version of the database installed.
3. With sample data ready and the latest database resources installed, create a simple dataflow that reads data from a file, processes it with the stage you want to optimize, and writes to a file. For example, if you want to test performance settings for Validate Address, create a dataflow consisting of Read from File, Validate Address, and Write to File.
4. Set the database resource pool size to 1:

   a. Open Management Console.
   b. Go to **Resources** > **Spectrum Databases**.
   c. Select the database resource you want to optimize and click the Modify button ✎.
   d. In the **Pool size** field, specify 1.
   e. Click **OK**.

5. Set the stage's runtime instances to 1:

   a. Open the dataflow in Enterprise Designer.
   b. Double-click the stage that you want to set to use multiple runtime instances.
   c. Click **Runtime**.

   > **Note:** Not all stages are capable of using multiple runtime instances. If there is no **Runtime** button at the bottom of the stage's window, the stage is not capable of using multiple runtime instances.

   d. Select **Local** and specify 1.
   e. Click **OK** to close the **Runtime Performance** window, then click **OK** to close the stage.

6. Calculate baseline performance by running the dataflow several times and recording the average values for:

   • Elapsed time
   • CPU utilization
   • Memory utilization

> **Tip:** You can use the JMX console to monitor performance. For more information, see **Monitoring Performance with the JMX Console**.

7. Run multiple instances of the job concurrently, if this is a use case that must be supported. Record elapsed time, CPU utilization, and memory utilization for each scenario.

> **Tip:** You can use a file monitor to run multiple instances of a job at once. For more information, see **Triggering a Flow with a Control File** on page 158.

8. Increment the database resource pool size and the stage runtime instances setting.
9. Restart the server.
10. Run the dataflow again, recording the elapsed time, CPU utilization, and memory utilization.
11. Continue to increment the database resource pool size and the stage runtime instances until you begin to see diminishing performance.
12. If you are testing geocoding performance, repeat this procedure using single country and multi-country input.

Distributed Processing

If you have a very complex job, or you are processing a very large data set such as one containing millions of records, you may be able to improve dataflow performance by distributing the processing of the dataflow to multiple instances of the Spectrum™ Technology Platform server on one or more physical servers.

The most scalable solution for distributed processing is to install Spectrum™ Technology Platform in a cluster. See the *Installation Guide* for instructions on installing and configuring a cluster.

> **Note:** While it is also possible to use distributed processing on a single Spectrum™ Technology Platform server, the following information describes using distributed processing in a cluster. If you are using a single server, distributed subflow processing is broken up into microbatches and processed by the one server instead of by the cluster.

Once your clustered environment is set up, you can build distributed processing into a dataflow by creating subflows for the parts of the dataflow that you want to distributed to multiple servers. Spectrum™ Technology Platform manages the distribution of processing automatically after you specify just a few configuration options for the subflow.

The following diagram illustrates distributed processing:

As records are read into the subflow, the data is grouped into batches. These batches are then written to the cluster and automatically distributed to the a node in the cluster which processes the batch. This processing is called a microflow. A subflow may be configured to allow multiple microflows to be processed simultaneously, potentially improving performance of the dataflow. When the distributed instance is finished processing a microflow, it sends the output back into the parent dataflow.

The more Spectrum™ Technology Platform nodes you have the more microflows can be processed simultaneously, allowing you to scale your environment as needed to obtain the performance you require.

Once set up, a clustered environment is easy to maintain since all nodes in the cluster automatically synchronize their configuration, which means the settings you apply through the Management Console and the dataflows you design in Enterprise Designer are available to all instances automatically.

Designing a Dataflow for Distributed Processing

Distributed processing takes parts of your dataflow and distributes the processing of those parts to a cluster of Spectrum™ Technology Platform servers. For example, your dataflow may perform

geocoding, and you might want to distribute the geocoding processing among several Spectrum™ Technology Platform nodes in a cluster to improve performance.

1. Decide which stages of your dataflow you want to distribute, then create a subflow containing the stages that you want to distribute.

   Do not use the following stages in a subflow that will be used for distributed processing:

   • Sorter
   • Unique ID Generator
   • Record Joiner
   • Interflow Match

   The following sets of stages must be used together in a subflow for distributed processing:

   • Matching stages (Intraflow Match and Transactional Match) and consolidation stages (Filter, Best of Breed and Duplicate Synchronization).
   • Aggregator and Splitter

   Do not include other subflows within the subflow (nested subflows).

   Note the following if you will be performing matching operations in a subflow used for distributed processing:

   • Sorting must be done in the job and not in the subflow. You must turn sort off in the stage and put the sort at job level.
   • Match Analysis is not supported in a distributed subflow
   • Collection numbers will be reused within a microflow batch group

   Using a Write Exception stage in a subflow may produce unexpected results. Instead, you should add this stage to your dataflow at the job level.

2. Once you have created your subflow for the portion of the dataflow you want to distribute, add the subflow to the parent dataflow and connect it to an upstream and downstream stage. Subflows used for distributed processing may have only one input port.

3. Right-click the subflow and select **Options**.

4. Select **Distributed**.

5. Enter the number of microflows to be sent to each server.

6. Enter the number of records that should be in each microflow batch.

7. Optional: (Optional) Check **Group field name** and select the name of the field by which the microflow batches should be grouped.

   If you provide a group field, your batch sizes could be greater than the number you specified in the **Micro flow batch size** field because a group will not be split across multiple batches. For example, if you specify a batch size of 100, but you have 108 records within the same group, that batch will include 108 records. Similarly, if you specify a batch size of 100, and a new group of 28 records with the same ID starts at record 80, you will have 108 records in that batch.

   The following example shows a dataflow where a subflow named My Distributed Subflow has been configured to run in distributed mode:

Running a Stage on a Remote Server

If your system administrator has enabled remote servers in Management Console, you can have stages in your dataflow execute their processing on a remote server. Using remote servers can improve performance by spreading dataflow processing among multiple Spectrum™ Technology Platform servers.

Your system administrator may have already designated certain stages to run on a remote server. If a stage is already routed to a remote server, you will see a red star in the top-left corner of the stage icon on the canvas in Enterprise Designer.

This procedure describes how to configure remote processing for a stage in a dataflow.

1. Open the dataflow in Enterprise Designer.
2. Double-click the stage you want to route to a remote server.
3. Click **Runtime**.

    The **Runtime Performance** dialog appears.

4. Click **Remote** and select the remote server to which you wish to route the process for this stage.
5. Click **OK**.

Troubleshooting Remote Server Errors

This section discusses possible errors you may experience when using remote servers.

Module Not Licensed

The remote server must have the license for both the module and the mode of execution you are trying to run, either batch or real-time. The license on the remote server may be different from the license on the local server. Log in to the remote server using Management Console and verify that the correct license is installed. You must log in with an account that has administrative privileges in order to view license information.

Remote Server Not Available

If the remote server is not running or is not reachable for any other reason, the remote services will become unavailable in Enterprise Designer and Management Console. You will see a yellow hazard icon in the status bar at the bottom of the screen:

Click this icon to see an error message that describes which remote servers are not available.

In addition, in Enterprise Designer any stages that use a remote stage will be replaced with an icon showing you the stage is no longer available:

Routing Has Changed

If you delete or undeploy a service that is installed both locally and remotely and has been routed through a remote server, and then click that service within Management Console, you will see a routing change indicator (a blinking exclamation point) next to the routing button on the Options tab for that service. This indicator means the routing has changed for that service.

Optimizing Stages

Optimizing Matching

Matching is typically one of the most time-consuming operations in any data quality implementation, making it important to ensure that matching is operating as efficiently as possible. There is always a balance between matching results and performance. If every record in a file is compared to every other record, you can be quite confident that all matches will be identified. However, this approach is unsustainable as the volume of data grows. For example, given an input file of 1 million records, matching each record to every other record would require nearly 1 trillion comparisons to evaluate each match rule.

Given that most records in a file do not match, the general approach to solving this problem is to define a match key and only compare those records that have the same match key. Proper match key definition is the most critical variable affecting performance of the matching engine. To define a proper match key, you must understand how the matching engine processes records and the options that are available.

The default matching method performs an exhaustive comparison of the record in a match queue to identify the maximum number of matches. Because of this, it is often the most time consuming way to do matching. Under the default matching method, the first record in the match queue becomes the suspect record. The next record is compared, and if it matches it is written out as a duplicate. If it does not match, it is added as a suspect, and the next record is compared to the two active suspects. Consider the following match queue:

Unique ID	Match Key
1	123A
2	123A
3	123A
4	123A
5	123A
6	123A
7	123A
8	123A
9	123A
10	123A

First, record 2 would be compared to record 1. Assuming it does not match, record 2 would be added as a suspect. Then record 3 would be compared to records 1 and 2, and so on. If there are no matching records, the total number of comparisons would be 45. If some records match, the number of comparisons will be less. For a match queue of a given size N, the maximum number of comparisons will be $N \times (N-1) \div 2$. When the queue size is small this is not noticeable, but as the queue size grows the impact is significant. For example, a queue size of 100 could result in 4,450 comparisons, and a queue size of 500 could result in 124,750 comparisons.

Defining an Appropriate Match Key

To define an appropriate match key, consider the following:

- The most important thing to remember is most records do not match. Therefore you want to compare only records that are likely to match.
- Only records with the same match key will be compared.

- Performance is a key consideration:

  - The match key determines the size of the match queue.
  - For a given number of records, as the match queue size doubles, execution time doubles.
  - A "tight" match key results in better performance. A "tight" match key is one that is specific, containing more characters from possibly more fields.
  - A "loose" match key may result in more matches. A "loose" match key is one that is less specific, containing fewer characters from possibly fewer fields.

Finding a Balance Between Performance and Match Results

To find a good balance between performance and results, consider the match rule and the density of the data.

- Consider the match rules:

  - Fields requiring an exact match could be included in the match key.
  - Build an appropriate key for the match rule. For example, for a phonetic match rule, a phonetic match key is probably appropriate.
  - A match key will often consist of parts of all the fields being matched.
  - Be aware of the effects of missing data.

- Consider the density of the data:

  - For example, in address matching, the match key would likely be tighter if all the records are in a single town instead of a national dataset.
  - Consider the largest match queue, not just the average. Review the Match Summary report to find the largest match queue.

- When using transactional match, the same considerations apply to the SELECT statement in Candidate Finder.

Express Match Key

In a typical file, most of the duplicate records match either exactly or nearly exactly. Defining an express match key allows the matching engine to perform an initial comparison of the express match keys to determine that two records are duplicates. This can significantly improve performance by avoiding the need to evaluate all the field level match rules.

Intraflow Match Methods

The default Intraflow Match match method compares all records having the same match key. For a match queue size of N, the default method performs anywhere from N−1 to N×(N−1) comparisons. If all records match, the number of comparisons is N−1. If no records match the number of comparisons is N×(N−1). Usually the number of comparisons is somewhere in the upper part of this range.

If performance is a priority, consider using the sliding window match method instead of the default method. The sliding window match method compares each record to the next W records (where W

is the window size). For a given file size N, the sliding window method performs no more than N×W comparisons. This can lead to better performance, but some matches may be missed.

Optimizing Candidate Finder

Candidate Finder selects candidate records from a database for comparison by Transactional Match. Since transactional match compares the suspect record to all of the candidate records returned by Candidate Finder, the performance of Transactional Match is proportional to the number of comparisons.

However, there are things you can do to improve the performance of Candidate Finder. To maximize the performance of Candidate Finder, a database administrator, or developer with extensive knowledge of the database schema and indexes, should tune the SQL SELECT statement in Candidate Finder. One of the most common performance problems is a query that contains a JOIN that requires a full table scan. In this case, consider adding an index or using a UNION instead of a JOIN. As a general rule, SQL queries should be examined and optimized by qualified individuals.

Optimizing Transforms

The Transformer stage provides a set of predefined operations that can be performed on the input data. Generally, these predefined transforms execute faster than custom transforms, since they are already compiled. However, when defining a large number of transforms, a custom transform will frequently execute faster. For example, to trim a number of fields, the following custom transform will typically execute faster than nine separate trim transforms.

```
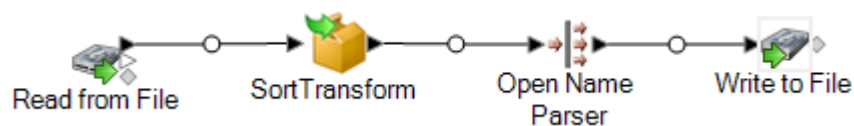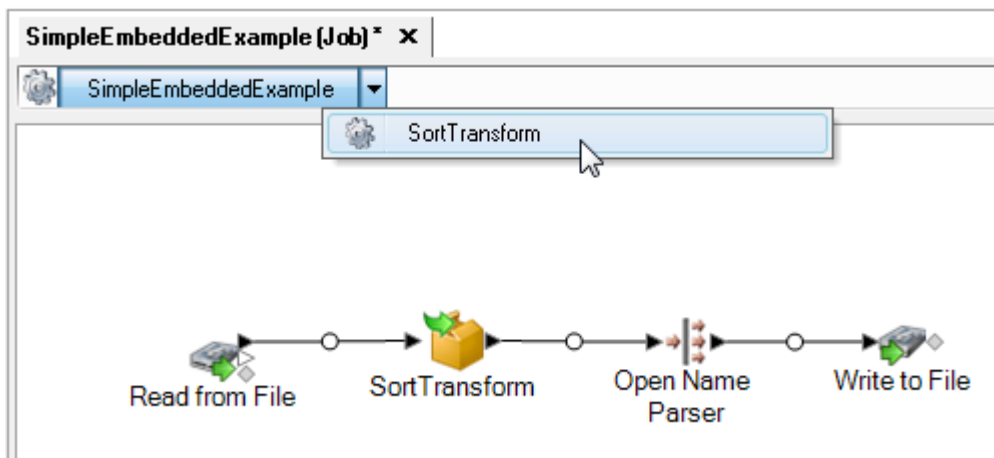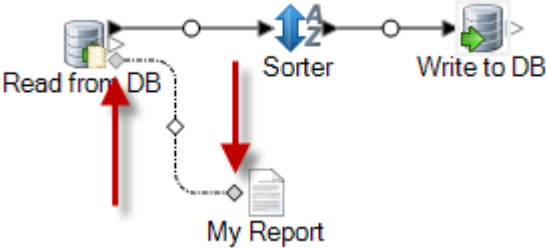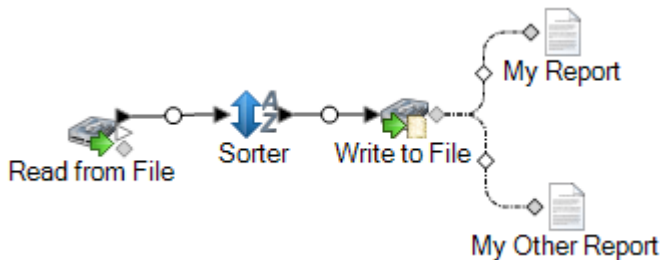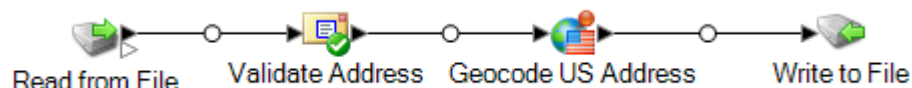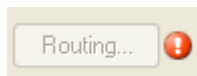data['AddressLine1'] = (data['AddressLine1'] != null) ?
data['AddressLine1'].trim() : null;
data['AddressLine2'] = (data['AddressLine2'] != null) ?
data['AddressLine2'].trim() : null;
data['AddressLine3'] = (data['AddressLine3'] != null) ?
data['AddressLine3'].trim() : null;
data['AddressLine4'] = (data['AddressLine4'] != null) ?
data['AddressLine4'].trim() : null;
data['City'] = (data['City'] != null) ? data['City'].trim() : null;
data['StateProvince'] = (data['StateProvince'] != null) ?
data['StateProvince'].trim() : null;
data['PostalCode'] = (data['PostalCode'] != null) ?
data['PostalCode'].trim() : null;
data['LastName'] = (data['LastName'] != null) ? data['LastName'].trim()
 : null;
data['FirstName'] = (data['FirstName'] != null) ?
data['FirstName'].trim() : null;
```

Optimizing Write to DB

By default the Write to DB stage commits after each row is inserted into the table. However, to improve performance enable the **Batch commit** option. When this option is enabled, a commit will be done after the specified number of records. Depending on the database this can significantly improve write performance.

When selecting a batch size, consider the following:

- **Data arrival rate to Write To DB stage:** If data is arriving at slower rate than the database can process then modifying batch size will not improve overall dataflow performance. For example, dataflows with address validation or geocoding may not benefit from an increased batch size.
- **Network traffic:** For slow networks, increasing batch size to a medium batch size (1,000 to 10,000) will result in better performance.
- **Database load and/or processing speed:** For databases with high processing power, increasing batch size will improve performance.
- **Multiple runtime instances:** If you use multiple runtime instances of the Write to DB stage, a large batch size will consume a lot of memory, so use a small or medium batch size (100 to 10,000).
- **Database roll backs:** Whenever a statement fails, the complete batch is rolled back. The larger the batch size, the longer it will take to perform the to rollback.

Optimizing Address Validation

Validate Address provides the best performance when the input records are sorted by postal code. This is because of the way the reference data is loaded in memory. Sorted input will sometimes perform several times faster than unsorted input. Since there will be some records that do not contain data in the postal code field, the following sort order is recommended:

1. Country (Only needed when processing records for multiple countries)
2. PostalCode
3. StateProvince
4. City

Optimizing Geocoding

Geocoding stages provide the best performance when the input records are sorted by postal code. This is because of the way the reference data is loaded in memory. Sorted input will sometimes perform several times faster than unsorted input. Since there will be some records that do not contain data in the postal code field, the following sort order is recommended:

1. PostalCode
2. StateProvince
3. City

You can also optimize geocoding stages by experimenting with different match modes. The match mode controls how the geocoding stage determines if a geocoding result is a close match. Consider consider setting the match mode to the **Relaxed** setting and seeing if the results meet your requirements. The **Relaxed** mode will generally perform better than other match modes.

Optimizing Geocode US Address

The Geocode US Address stage has several options that affect performance. These options are in this file:

SpectrumLocation\server\modules\geostan\java.properties

egm.us.multimatch.max.records	Specifies the maximum number of matches to return. A smaller number results in better performance, but at the expense of matches.
egm.us.multimatch.max.processing	Specifies the number of searches to perform. A smaller number results in better performance, but at the expense of matches.
FileMemoryLimit	Controls how much of the reference data is initially loaded into memory.

Dataflow Versions

The Versions feature in Enterprise Designer allows you to keep a revision history of your dataflows. You can view previous versions of a dataflow, expose older versions for execution, and keep a history of your changes in case you ever need to revert to a previous version of a dataflow.

Saving a Dataflow Version

There are two ways to save a version of your dataflow in Enterprise Designer:

- Expose your dataflow. Each time you expose a dataflow, either by selecting **File** > **Expose/Unexpose and save** or by clicking the light bulb in the tool bar, Enterprise Designer automatically saves a version of the dataflow.
- Manually save a version in the **Versions** pane in Enterprise Designer.

> **Note:** A dataflow version is not created when you simply save a dataflow.

The following procedure describes how to manually save a version in the **Versions** pane of Enterprise Designer.

1. In Enterprise Designer, open the dataflow.
2. If the **Versions** pane is not visible, select **View** > **Versions**
3. Make sure that the latest saved version is selected in the **Versions** list. This is the version at the top of the list.
4. Click the green plus icon in the **Versions** pane.

A new version of the dataflow is saved and added to the **Versions** pane.

Viewing a Dataflow Version

You can view a previous version of a dataflow. This allows you to see how a dataflow was designed in the past before more recent changes were made. Previous versions can only be viewed, not modified. In order to modify a previous version it must first be promoted to the latest saved version.

1. In Enterprise Designer, open the dataflow.
2. If the **Versions** pane is not visible, select **View** > **Versions**
3. Select the version that you want to view.

The selected version is displayed on the dataflow canvas.

Editing a Dataflow Version

You can edit a previous version of a dataflow by promoting it to the latest-saved version. Promoting a dataflow version moves it to the latest-saved version, making it available for editing.

> **Note:** Before performing this procedure, note that the existing latest-saved version will be overwritten by the version you promote and edit. If you want to preserve a copy of the existing latest-saved version, save it as a version before promoting the older version.

1. In Enterprise Designer, open the dataflow.
2. If the **Versions** pane is not visible, select **View** > **Versions**
3. Select the version that you want to edit.
4. Click the promote icon.

The selected version is promoted to the latest-saved version. You can now edit the dataflow.

Editing Version Properties

When you save a dataflow version, it is given a default version number. You can modify the version number and add comments to document the version's changes or purpose.

1. In Enterprise Designer, open the dataflow.
2. If the **Versions** pane is not visible, select **View** > **Versions**
3. Select the version that you want to modify.

4. Click the properties icon:

5. In the **Name** field, enter a name for the version. You can use version numbers or any meaningful name. The name can be anything you choose.

6. In the **Comment** field, you can enter a longer comment that describes in more detail the purpose of the version of the changes you made. Adding a comment is optional.

7. Click **OK**.

Exposing a Version

If you have saved multiple versions of a dataflow you can choose which version to expose for execution.

1. In Enterprise Designer, open the dataflow.
2. If the **Versions** pane is not visible, select **View** > **Versions**
3. In the **Versions** pane, select the version of the dataflow that you want to expose.
4. Select **File** > **Expose/Unexpose and Save**

The selected version is now exposed and available for execution. The version with the light bulb next to it is the version that is exposed, as shown here:

When a dataflow is exposed the light bulb button in the Enterprise Designer tool bar indicates that the dataflow is exposed as shown here:

The light bulb indicates that the dataflow is exposed even if you are viewing a version other than the exposed version. If you click the light bulb while viewing an unexposed version it will switch the exposed version to the version you are currently viewing. If you click the light bulb while viewing the exposed version, it will unexpose the dataflow.

3 - Inspecting and Testing

In this section

Checking a Flow for Errors

Enterprise Designer automatically checks a flow for errors when you run a flow, run inspection, expose a flow, or save an exposed flow. You can also check for errors by clicking the validation button ✅.

When an error is found, the **Validation** pane appears at the bottom of the Enterprise Designer window. Click an error to highlight the error on the canvas. Double-click an error to open the options window of the item containing the error.

Inspecting a Dataflow

To view the effect of your dataflow on the input data at different points in the dataflow, use the inspection tool in Enterprise Designer. Inspection enables you to confirm that the dataflow is having the desired effect on your data, isolate problems, or identify records that contain defects.

1. Specify the data to use for inspection.

   The data should be representative of actual data, or, if you are troubleshooting a specific issue, should be the data that causes the issue you are troubleshooting. There are two ways to specify the data to use for inspection, depending on whether you are inspecting a service or a job.

Scenario	Description
To specify inspection data for a job	When inspecting a job, the data used for inspection is the data specified in the source stage. The inspection tool can process a maximum of 50 records, which by default is the first 50 records in the input file or database. If you want to use data that starts somewhere other than the first record, double-click the Read From File stage and complete the **Starting record** field in the Runtime tab.
To specify inspection data for a service	Service dataflows use an Input stage to define the input to the dataflow. Because an Input stage does not have access to data when you are editing the dataflow, you must define inspection data in the Input stage on the **Inspection Data** tab. You can specify a maximum of 50 records.

   There are a few ways you can enter inspection data in an Input stage.

   - If you want to use just a few records for inspection, you can manually type in the data.

     Tip: If want to save the inspection data you enter to use again in another stage, you can export the inspection data to a text file by clicking **Export Data**.

   - If you have data in a CSV or TXT file, you can import the data by clicking **Import Data**. The data must use one of these delimiters:

     - \t
     - |
     - ,
     - ;

   - You can copy delimited data from another application and paste it into the inspection data editor.

Scenario	Description
	The **Inspection Input** tab indicates pass-through data by enclosing the field name in parentheses, as shown here:

Note: Certain field types have restrictions when used for inspection:

- Double and float fields must contain numeric data only. The field may have up to 16 digits and 6 decimal places. Exponential notation is not supported in inspection.
- Integer and long fields must contain numeric data only.

2. Indicate the points in the dataflow where you want to view data.

Scenario	Description
To add an inspection point to a channel	Right-click to the left of the Rename node on a channel and select **Add Inspection Point**.

A point is added to the job:

Scenario	Description

To compare records at two points in a dataflow	Add two inspection points at the points in the dataflow that you want to compare:

Tip: If you are using inspection to identify a problem, inspect outer points on the dataflow first then move inward to narrow down where a problem may be.

To inspect a subflow embedded in a job or service	Right-click the subflow stage and select **Inspect this Dataflow**:

The input data (in a job) or the inspection data (in a service) is automatically passed to the subflow, so there is no need to enter inspection data in the subflow's Input stage.

Scenario	Description

> **Note:** When you inspect a subflow, the exposed version of the subflow is shown. If make a change to the subflow and want to re-run inspection, you need to expose the new version.

3. Select **Run** > **Inspect Current Flow** or click the **Inspect Current Flow** button on the toolbar.

If you specified one inspection point, the **Inspection Results** pane shows the inspected data in horizontal view. You can change the layout of the view using the toolbar icons above the table. If your inspection data is hierarchical, it cannot be viewed vertically.

> **Note:** Date and time data is displayed in the format specified in the type conversion options.

> **Tip:** You can move an inspection point by dragging it to another channel. The inspection data updates automatically.

If you specified two inspection points, the **Inspection Results** pane displays the records as they exist at the two points. The left pane shows the left-most inspection point in the dataflow and the right pane shows the right-most inspection point in the dataflow. Click a record in the right pane to highlight the corresponding record in the left pane to see how the record has changed between the two inspection points.

Each column represents a field in the dataflow. Columns are arranged in alphabetical order. New fields added between the inspection points are shown in the right pane after the original columns. To reorder columns, click and drag them into the order you want.

These situations influence how the inspection results are displayed for two inspection points:

- If there is a Sorter stage between the two inspection points, the records in the inspection results will be sorted as they were before the Sorter stage. Sorting is ignored in the second inspection point so that you can compare corresponding records from each inspection point side by side.
- If there are stages between the two inspection points that create new records, such as an Aggregator stage, the records shown in the second inspection point will not have a corresponding record in the first inspection point.
- Records that exist at the second inspection point but not at the first are displayed at the bottom of the list of records in the second inspection point.

4. When you update or make changes to the dataflow, click **Run** > **Inspect Current Flow** to refresh the inspection results.

5. When you close the Inspection Results pane, the inspection data is lost. Similarly, when you close a job, the inspection points and inspection data are lost. To save the inspection results to a file:

   a) In the inspection results grid, select the rows you wish to save. You can select all data by right-clicking in either pane and clicking **Select All**.

   b) Select **Copy** from the context menu.

   c) Open the application into which you want to save the data (for example, Microsoft Excel or Notepad).

   d) In the application, paste the data.

   e) Save the file.

Testing a Service with Management Console

Management Console provides a preview feature which allows you to send test data to a service and see the results.

1. In a web browser go to this URL:

   http://*server*:*port*/managementconsole

   Where *server* is the server name or IP address of your Spectrum™ Technology Platform server and *port* is the HTTP port used bySpectrum™ Technology Platform. By default, the HTTP port is 8080.

2. Go to the **Services** menu and click the module containing the service you want to test.

3. Click the service you want to test.

4. Click **Preview**.

5. Enter the input data you want to use for your test. To import data from a file, click the Import button ⬇.

6. Click **Run Preview**.

4 - Running a Flow

In this section

Running a Job or Process Flow

Running a Dataflow Flow in Enterprise Designer

The following procedure describes how to manually run a job or process flow.

1. In Enterprise Designer, select **File** > **Open** and open the dataflow you want to run.
2. Validate a dataflow prior to running it to ensure that it contains no errors. To validate a dataflow, select **Run** > **Validate**.
3. Select **Run** > **Run current flow**.

Running A Job from the Command Line

Before you can run a job from the command line, it must be exposed. To expose a job, open the job in Enterprise Designer and select **File** > **Expose/Unexpose and Save**.

To run a job from the command line, you must install the job executor utility on the system where you want to run the job. The Job Executor is available from the Spectrum™ Technology Platform Welcome page on the Spectrum™ Technology Platform server (for example, http://myserver:8080).

Usage
```
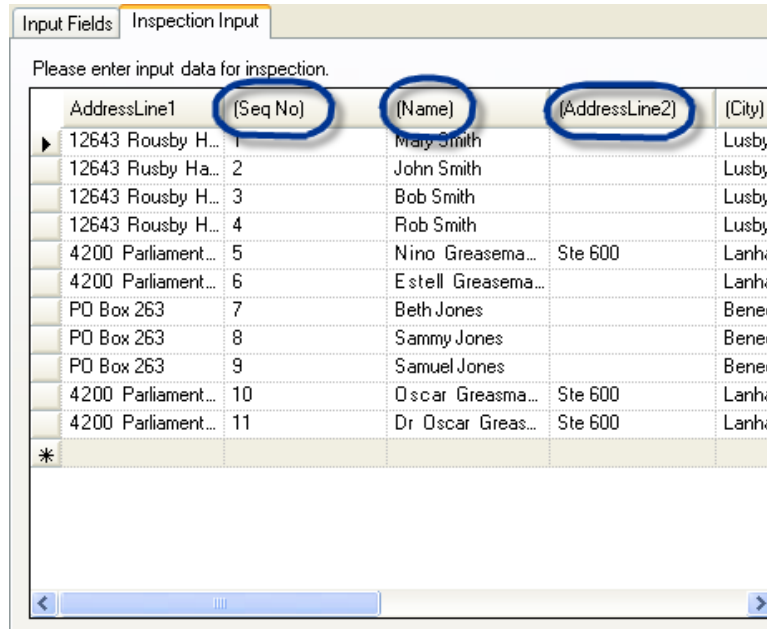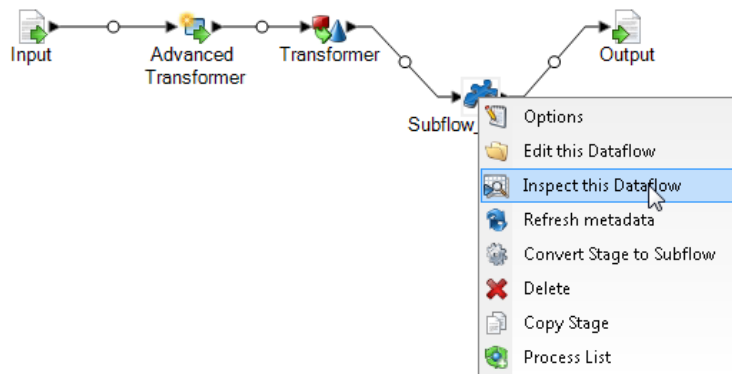java -jar jobexecutor.jar -u UserID -p Password -j Job [Optional Arguments]
```

Required	Argument	Description
No	-?	Prints usage information.
No	-d *delimiter*	Sets instance/status delimiter. This appears in synchronous output only.
No	-e	Use a secure HTTPS connection for communication with the Spectrum™ Technology Platform server.
No	-f *property file*	Specifies a path to a job property file. A job property file contains job executor arguments. For more information on job property files, see **Using a Job Property File** on page 152.
No	-h *host name*	Specifies the name or IP address of the Spectrum™ Technology Platform server.

Required	Argument	Description
No	`-i` *poll interval*	Specifies how often to check for completed jobs, in seconds. This applies only in synchronous mode.
Yes	`-j` *job name*	A comma-separated list of jobs to run. Job names are case-sensitive. Jobs are started in the order listed.
No	`-n` *email list*	Specifies a comma-separated list of additional email addresses for configured job notifications.
No	`-o` *property file*	Specifies a path to a dataflow options property file. Use a dataflow options property file to set options for stages in the dataflow. In order to set dataflow options using a property file, you must configure the dataflow to expose stage options at runtime. For more information, see **Adding Dataflow Runtime Options** on page 168. For example, a dataflow options properties file for a dataflow that contains an Assign GeoTAX Info stage may look like this: ```OutputCasing=U\nUseStreetLevelMatching=N\nTaxKey=T\nDatabase.GTX=gsl```
Yes	`-p` *password*	The password of the user.
No	`-r`	Specify this argument to return a detailed report about the job. This option only works if you also specify `-w` . The report contains the following information: • **Position 1**—Name of job • **Position 2**—Job process ID • **Position 3**—Status • **Position 4**—Start Date/Time (MM/DD/YYYY HH:MM:SS) • **Position 5**—End Date/Time (MM/DD/YYYY HH:MM:SS) • **Position 6**—Number of successful records • **Position 7**—Number of failed records • **Position 8**—Number of malformed records

Required	Argument	Description
		• **Position 9**—Currently unused
		For example:
		`MySimpleJob|4|succeeded|04/09/2010 14:50:47|04/09/2010 14:50:47|100|0|0|`
		The information is delimited using the delimiter specified in the `-d` argument.
No	`-s` *port*	The socket (port) on which the Spectrum™ Technology Platform server is running. The default is 8080.
No	`-t` *timeout*	Sets the timeout (in seconds) for synchronous mode. The default is 3600. The maximum is 2147483. This is a global, aggregate timeout and represents the maximum time to wait for all spawned jobs to complete.
Yes	`-u` *user name*	The login name of the user.
No	`-v`	Return verbose output.
No	`-w`	Runs job executor in synchronous mode. This means that job executor remains running until the job completes.
		If you do not specify `-w`, job executor exits after starting the job, unless the job reads from or writes to files on the server. In this case, job executor will run until all local files are processed, then exit.
No	*StageName=Protocol*:*FileName*	Overrides the input or output file specified in Read from File or Write to File. For more information, see **Overriding Job File Locations** on page 148.
No	*StageName*:schema=*Protocol*:*SchemaFile*	Overrides the file layout definition specified in Read from File or Write to File with one defined in a schema file. For more information, see **Overriding the File Format at the Command Line** on page 150.

Example Use of Job Executor

The following example shows command line invocation and output:

```
D:\spectrum\job-executor>java -jar jobexecutor.jar -u user123
 -p "mypassword" -j validateAddressJob1 -h spectrum.example.com
 -s 8888 -w -d "%" -i 1 -t 9999

validateAddressJob1%105%succeeded
```

In this example, the output indicates that the job named 'validateAddressJob1' ran (with identifier 105) with no errors. Other possible results include "failed" or "running."

Overriding Job File Locations

When you run a job at the command line using job executor or the Administration Utility, you can override the input file specified in the dataflow's source stage (such as Read from File), as well as the output file specified in the dataflow's sink stage (such as Write to File).

To do this in job executor, specify the following at the end of the job executor command line command:

StageName=*Protocol*:*FileName*

In the Administration Utility, use the `--l` argument in the `job execute` command:

`--l` *StageName*=*Protocol*:*FileName*

Where:

StageName

> The stage label shown under the stage's icon in the dataflow in Enterprise Designer. For example, if the stage is labeled "Read from File" you would specify `Read from File` for the stage name.
>
> To specify a stage within an embedded dataflow or a subflow, preface the stage name with the name of the embedded dataflow or subflow, followed by a period then the stage name:
>
> *EmbeddedOrSubflowName*.*StageName*
>
> For example, to specify a stage named Write to File in a subflow named Subflow1, you would specify:
>
> `Subflow1.Write to File`
>
> To specify a stage in an embedded dataflow that is within another embedded dataflow, add the parent dataflow, separating each with a period. For example, if Embedded Dataflow 2 is inside Embedded Dataflow 1, and you want to specify the Write to File stage in Embedded Dataflow 2, you would specify this:
>
> `Embedded Dataflow 1.Embedded Dataflow 2.Write to File`

Protocol

> A communication protocol. One of the following:

file   Use the file protocol if the file is on the same machine as the Spectrum™ Technology Platform server. For example, on Windows specify:

`"file:C:/myfile.txt"`

On Unix or Linux specify:

`"file:/testfiles/myfile.txt"`

esclient   Use the esclient protocol if the file is on the computer where you are executing the job if it is a different computer from the one running the Spectrum™ Technology Platform server. Use the following format:

`esclient:`*`ComputerName`*`/`*`path to file`*

For example,

`esclient:mycomputer/testfiles/myfile.txt`

> **Note:** If you are executing the job on the server itself, you can use either the file or esclient protocol, but are likely to have better performance using the file protocol.

If the host name of the Spectrum™ Technology Platform server cannot be resolved, you may get the error "Error occurred accessing file". To resolve this issue, open this file on the server: *SpectrumLocation*`/server/app/conf/spectrum-container.properties`. Set the `spectrum.runtime.hostname` property to the IP address of the server.

esfile   Use the esfile protocol if the file is on a file server. The file server must be defined in Management Console as a resource. Use the following format:

`esfile://`*`file server`*`/`*`path to file`*

For example,

`esfile://myserver/testfiles/myfile.txt`

Where myserver is an FTP file server resource defined in Management Console.

webhdfs   Use the webhdfs protocol if the file is on a Hadoop Distributed File Server. The HDFS server must be defined in Management Console as a resource. Use the following format:

`webhdfs://`*`file server`*`/`*`path to file`*

For example,

`webhdfs://myserver/testfiles/myfile.txt`

Where myserver is an HDFS file server resource defined in Management Console.

FileName

The full path to the file you want to use as input or output.

Note: You must use forward slashes (/) in file paths, not backslashes.

To specify multiple overrides, separate each override with a comma.

Example File Override

The following job executor command would use the file `C:/myfile_input.txt` as the input file for the Read from File stage and would use the file `C:/myfile_output.txt` as the output file for the Write to File stage.

```
java -jar jobexecutor.jar -j Job1 -u Bob1234 -p "" "Read from
File"="file:C:/myfile_input.txt" "Write to
File"="file:C:/myfile_output.txt"
```

Overriding the File Format at the Command Line

When you run a job using job executor or the Administration Utility, you can override the file layout (or schema) of the file specified in the dataflow's Read from File stage and Write to File stage.

To do this in job executor, specify the following at the end of the job executor command line command:

`StageName:schema=Protocol:SchemaFile`

In the Administration Utility, use the `--l` argument in the `job execute` command:

`--l StageName:schema=Protocol:SchemaFile`

Where:

StageName

The stage label shown under the stage's icon in the dataflow in Enterprise Designer. For example, if the stage is labeled "Read from File" you would specify `Read from File` for the stage name.

To specify a stage within an embedded dataflow or a subflow, preface the stage name with the name of the embedded dataflow or subflow, followed by a period then the stage name:

`EmbeddedOrSubflowName.StageName`

For example, to specify a stage named Write to File in a subflow named Subflow1, you would specify:

`Subflow1.Write to File`

To specify a stage in an embedded dataflow that is within another embedded dataflow, add the parent dataflow, separating each with a period. For example, if Embedded Dataflow 2 is inside Embedded Dataflow 1, and you want to specify the Write to File stage in Embedded Dataflow 2, you would specify this:

`Embedded Dataflow 1.Embedded Dataflow 2.Write to File`

Protocol

A communication protocol. One of the following:

file
Use the file protocol if the file is on the same machine as the Spectrum™ Technology Platform server. For example, on Windows specify:

```
"file:C:/myfile.txt"
```

On Unix or Linux specify:

```
"file:/testfiles/myfile.txt"
```

esclient
Use the esclient protocol if the file is on the computer where you are executing the job if it is a different computer from the one running the Spectrum™ Technology Platform server. Use the following format:

```
esclient:ComputerName/path to file
```

For example,

```
esclient:mycomputer/testfiles/myfile.txt
```

> **Note:** If you are executing the job on the server itself, you can use either the file or esclient protocol, but are likely to have better performance using the file protocol.

If the host name of the Spectrum™ Technology Platform server cannot be resolved, you may get the error "Error occurred accessing file". To resolve this issue, open this file on the server: *SpectrumLocation*/server/app/conf/spectrum-container.properties. Set the `spectrum.runtime.hostname` property to the IP address of the server.

esfile
Use the esfile protocol if the file is on a file server. The file server must be defined in Management Console as a resource. Use the following format:

```
esfile://file server/path to file
```

For example,

```
esfile://myserver/testfiles/myfile.txt
```

Where myserver is an FTP file server resource defined in Management Console.

webhdfs
Use the webhdfs protocol if the file is on a Hadoop Distributed File Server. The HDFS server must be defined in Management Console as a resource. Use the following format:

```
webhdfs://file server/path to file
```

For example,

```
webhdfs://myserver/testfiles/myfile.txt
```

Where myserver is an HDFS file server resource defined in Management Console.

SchemaFile

The full path to the file that defines the layout you want to use.

> **Note:** You must use forward slashes (/) in file paths, not backslashes.

To create a schema file, define the layout you want in Read from File or Write to File, then click the **Export** button to create an XML file that defines the layout.

> **Note:** You cannot override a field's data type in a schema file when using job executor. The value in the <Type> element, which is a child of the <FieldSchema> element, must match the field's type specified in the dataflow's Read from File or Write to File stage.

Example File Format Override

The following job executor command would use the file `C:/myschema.xml` as the layout definition for the file read in by the Read from File stage.

```
java -jar jobexecutor.jar -j Job1 -u Bob1234 -p "" "Read from
File":schema="file:C:/myschema.xml"
```

Using a Job Property File

A job property file contains arguments that control the execution of jobs when you use the job executor or the Administration Utility to run a job. Use a job property file if you want to reuse arguments by specifying a single argument at the command line (`-f`) rather than specifying each argument individually at the command line.

To create a property file, create a text file with one argument on each line. For example:

```
d %
h spectrum.mydomain.com
i 30
j validateAddressJob1
u user
p password
s 8888
t 9999
w true
```

The job property file can contain these arguments:

Required	Argument	Description
No	?	Prints usage information.
No	d *delimiter*	Sets instance/status delimiter. This appears in synchronous output only.
No	e	Use a secure HTTPS connection for communication with the Spectrum™ Technology Platform server.

Required	Argument	Description
No	`h` *hostname*	Specifies the name or IP address of the Spectrum™ Technology Platform server.
No	`i` *pollinterval*	Specifies how often to check for completed jobs, in seconds. This applies only in synchronous mode.
Yes	`j` *jobname*	A comma-separated list of jobs to run. Job names are case-sensitive. Jobs are started in the order listed.
No	`n` *emaillist*	Specifies a comma-separated list of additional email addresses for configured job notifications.
Yes	`p` *password*	The password of the user.

No	`r`	Returns a delimited list with the following information about the job written to standard output:
		
		• **Position 1**—Name of job
		• **Position 2**—Job process ID
		• **Position 3**—Status
		• **Position 4**—Start Date/Time (MM/DD/YYYY HH:MM:SS)
		• **Position 5**—End Date/Time (MM/DD/YYYY HH:MM:SS)
		• **Position 6**—Number of successful records
		• **Position 7**—Number of failed records
		• **Position 8**—Number of malformed records
		• **Position 9**—Currently unused
		
		The information is delimited using the delimiter specified in the -d argument. For example:
		
		`MySimpleJob|4|succeeded|04/09/2010`
		`14:50:47|04/09/2010 14:50:47|100|0|0|`

No	`s` *port*	The socket (port) on which the Spectrum™ Technology Platform server is running. The default is 8080.
No	`t` *timeout*	Sets the timeout (in seconds) for synchronous mode. The default is 3600. The maximum is 2147483. This is a global, aggregate timeout and represents the maximum time to wait for all spawned jobs to complete.
Yes	`u` *username*	The login name of the user.
No	`v`	Return verbose output.
No	`w`	Specifies to wait for jobs to complete in a synchronous mode.

Using Both Command Line Arguments and a Property File

A combination of both command-line entry and property file entry is also valid. For example:

```
java -jar jobexecutor.jar -f /dcg/job.properties -j job1
```

In this case command line arguments take precedence over arguments specified in the properties file. In the above example, the job job1 would take precedence over a job specified in the properties file.

Running a Process Flow from the Command Line

To run a process flow from the command line, use the Process Flow Executor. You can install the Process Flow Executor from the Spectrum™ Technology Platform Welcome page (for example, http://myserver:8080).

> **Note:** You can also use the Administration Utility to execute process flows from the command line.

Usage

```
java -jar pflowexecutor.jar -r ProcessFlowName -u UserID -p Password [Optional
Arguments]
```

Required	Argument	Description
No	-?	Prints usage information.
No	-d DelimiterCharacter	Sets a delimiter to use to separate the status information displayed in the command line when you execute the command. The default is "|". For example, using the default character, the following would be displayed at the command line when you execute a processflow named "MyProcessflow": MyProcessflow|1|Succeeded
No	-e	Use an HTTPS connection for communication with the Spectrum™ Technology Platform server. **Note:** If you specify any file overrides this argument must not be the last argument specified.
No	-f PropertyFile	Specifies a path to a property file. For more information on property files, see **Using a Process Flow Property File** on page 156.
No	-h HostName	Specifies the name or IP address of the Spectrum™ Technology Platform server.
No	-i PollInterval	Specifies how often to check for completed jobs, in seconds. The default is "5".
Yes	-p Password	The password of the user. Required.

Required	Argument	Description
Yes	-r *ProcessFlowNames*	A comma-separated list of process flows to run. Required. **Note:** If you specify any file overrides this argument must not be the last argument specified.
No	-s *Port*	The socket (port) on which the Spectrum™ Technology Platform server is running. The default is 8080.
No	-t *Timeout*	This option is deprecated and will be ignored.
Yes	-u *UserName*	The login name of the user. Required.
No	-v *Verbose*	Return verbose output where *Verbose* is one of the following: **true** Return verbose output. **false** Do not return verbose output. **Note:** If you specify any file overrides this argument must not be the last argument specified.
No	-w *WaitToComplete*	This option is deprecated and will be ignored.
No	*StageName=FileName*	Overrides the input or output file specified in the job. For more information, see **Overriding Process Flow File Locations**.

Examples

This is a basic command-line entry, with a process flow name and user ID, and password:

```
java -jar pflowexecutor.jar -r MyFlow1 -u Bob1234 -p
"mypassword1"
```

This example shows the same information as above but with additional arguments:

```
java -jar pflowexecutor.jar -r Flow1 -u Bob1234 -p
"mypassword1" -h spectrum.example.com -s 8888 -w -d "%" -i
1 -t 9999
```

The following example shows command line invocation and output.

```
D:\spectrum\pflow-executor>java -jar pflowexecutor.jar -u
Bob1234 -p "mypassword1" -r
validateAddressFlow1 -h spectrum.example.com -s 8888 -w -d
"%" -i
1 -t 9999
validateAddressJob1%111%succeeded
```

In this example, the process flow named validateAddressFlow1 ran (with identifier 111). No errors occurred. Other possible results include "failed" or "running."

Using a Process Flow Property File

A property file contains arguments that you can reuse by specifying the path to the property file with the `-f` argument in the process flow executor. The property file must contain, at minimum, the process flow (`r`), user ID (`u`), and password (`p`).

1. Open a text editor.
2. Specify one argument on each line as shown in the following example. See **Running a Process Flow from the Command Line** on page 154 for a list of arguments.

    > **Note:** You cannot use a property file to override input or output files. Overriding input and output files can only be done using command line arguments.

    ```
    d=%
    h=myserver.mydomain.com
    i=30
    u=user
    p=password
    r=MyFlow1
    s=8888
    ```

3. Save the file with a file extension of .properties (for example, "example.properties").
4. When you run the process flow executor, specify the path to the property file using the `-f` argument. A combination of both command-line entry and property file entry is also valid. Command line arguments take precedence over arguments specified in the properties file.
   `java -jar pflowexecutor.jar -f /dcg/flow.properties -r MyFlow2`

   In the above example, the process flow MyFlow2 would take precedence over a process flow specified in the properties file.

Scheduling a Dataflow

A dataflow schedule runs a job or process flow automatically at a specific time. You can schedule a dataflow to run once, or set up recurring execution.

1. If you have not already done so, expose the job or process flow.

   You can expose jobs and process flows by opening the job or process flow in Enterprise Designer and selecting **File** > **Expose/Unexpose and Save**.

2. Open the Management Console.

3. Browse to **Execution** then click **Scheduling**.

4. Click **Add** to create a new schedule or, if you want to modify an existing schedule, choose the schedule and click **Modify**.

5. In the **Add Task** or **Modify Task** window, choose the settings for this task.

   - **Task Name**—The name you want to give to this scheduled task. This is the name that will be displayed in the task listing.
   - **Flow type**—Choose the type of process you are scheduling, either a job or a process flow.
   - **Flow name**—Select the job or process flow that you want to schedule. Only jobs and process flows that are saved and exposed are available here. If the job or process flow that you want is not shown, open the job or process flow in Enterprise Designer then select **File** > **Expose/Unexpose and Save**.
   - **Enable task**—Check this box to run the job or process flow at the specified time. Clear this box to suspend the schedule.
   - **Schedule**—Specify the date and time you want the job or process flow to run.

6. If the flow uses files for input or output, those files must reside on the Spectrum™ Technology Platform server or on a file server defined as an external resource in Management Console. This applies both to jobs as well as job activities within a process flow. If a source or sink stage references a file on a client computer do one of the following:

Option	Description
Option 1: Modify the dataflow	Move the file to the Spectrum™ Technology Platform server or file server then modify the dataflow: 1. Open the flow in Enterprise Designer. 2. Double-click the source or sink stage. 3. In the **File name** field, click the browse button. 4. Click **Remote Machine** then select the file you want. **Note:** If you are running Enterprise Designer on the same machine as the Spectrum™ Technology Platform server, it will appear that clicking Remote Machine is no different than clicking My Computer. However, you must select the file using Remote Machine in order for the system to recognize the file as being on the Spectrum™ Technology Platform server.
Option 2: Override the dataflow file location when this schedule runs	You can override the input file specified in the dataflow's source stage (such as Read from File), as well as the output file specified in the dataflow's sink stage (such as Write to File). 1. Click **Options**.

Option	Description
	2. Under **Stage file locations** select the stage that references a local file.
	3. Click **Modify** and select the file on the Spectrum™ Technology Platform server.

7. If you want the job or process flow to run on a recurring schedule, check the **Task recurrence** check box then click the **Recurrence** button and complete the fields.

8. If the dataflow has been configured for email notification, you can specify additional recipients for the notifications that will be sent when the dataflow runs.

   a) Click **Options**.
   b) Under Notification, click **Add**.
   c) Enter the email address you want the notification to be sent to. For example, me@mycompany.com.
   d) Click **OK**.

      Note: Notification must be configured in Management Console in order for email notifications to work. In addition, verify that the dataflow has been configured to support notification. To do this, open the dataflow in Enterprise Designer, select **Edit** > **Notifications**.

9. Click **OK**.

Triggering a Flow with a Control File

A flow can run automatically when a control file is detected in a monitored directory. This feature is useful in situations where the flow needs another process to complete before running. For example, you may have a flow that needs an input file generated by another business process. You can set up the other process to place a control file into a folder, and configure Spectrum™ Technology Platform to run a flow when that control file appears.

   Note: Be sure that the control file is placed in the monitored folder only after all files required by the flow are in place and ready for processing.

1. If you have not already done so, expose the flow.

   You can expose a flow by opening it in Enterprise Designer and selecting **File** > **Expose/Unexpose and Save**.

2. Open Management Console.
3. Go to **Flows** > **Schedules**.
4. Click the Add button ✚ .

5. In the **Name** field, enter the name you want to give to this schedule. This is the name that will be displayed in the schedules listing.

6. In the **Flow** field, enter the job or process flow that you want to run. Only jobs and process flows that are saved and exposed are available here.

7. After you specify a flow, additional fields appear below the **Flow** field, one field for each of the flow's source stages (such as Read from File) and sink stages (such as Write to File). These fields show the files that will be used when the flow is executed by this schedule. By default, the flow will use the files specified in the flow's sources and sinks. You can specify different files to use when this schedule runs by replacing the file path with the path to another file. For example, if your flow has a Read from File stage that reads data from `C:\FlowInput\Customers.csv` but you want to use data from `C:\FlowInput\UpdatedCustomers.csv` when this schedule runs, you would specify `C:\FlowInput\UpdatedCustomers.csv` in the Read from File field.

    > **Note:** In order change the files used in the source and sink stages you must have Read permission for the **Resources - File Servers** secured entity type.

    Note that when a flow is triggered by a schedule the files used by a flow must reside on the Spectrum™ Technology Platform server or on a file server defined as an external resource in Management Console. This applies both to jobs as well as job activities within a process flow. If a source or sink stage references a file on a client computer do one of the following:

Option	Description
Option 1: Modify the dataflow	Move the file to the Spectrum™ Technology Platform server or file server then modify the dataflow: 1. Open the dataflow in Enterprise Designer. 2. Double-click the source or sink stage. 3. In the **File name** field, click the browse button. 4. Click **Remote Machine** then select the file you want. > **Note:** If you are running Enterprise Designer on the same machine as the Spectrum™ Technology Platform server, it will appear that clicking Remote Machine is no different than clicking My Computer. However, you must select the file using Remote Machine in order for the system to recognize the file as being on the Spectrum™ Technology Platform server.
Option 2: Override the dataflow file location when this schedule runs	You can override the file references contained in the flow when this schedule runs. To do this, replace the default file shown in each source and sink field with a path to a file on the Spectrum™ Technology

Option	Description
	Platform server or a file server resource defined in Management Console.

8. In the **Trigger** field, choose **Control File**.

9. In the **Control file** field, specify the full path and name of the control file that will trigger the flow. You can specify an exact file name or you can use the asterisk (*) as a wild card. For example, `*.trg` would trigger the flow when any file with a `.trg` extension appears in the folder.

   The presence of a control file indicates that all files required for the flow are in place and ready to be used in the flow.

   The control file can be a blank file. For jobs, the control file can specify overrides to file paths configured in the Write to File or Read from File stages. To use a control file to override the file paths, specify the Read from File or Write from File stage names along with the input or output file as the last arguments like this:

   ```
   stagename=filename
   ```

   For example:

   ```
   Read\ from\ File=file:C:/myfile_input.txt
   Write\ to\ File=file:C:/myfile_output.txt
   ```

   The stage name specified in the control file must match the stage label shown under the stage's icon in the dataflow. For example, if the input stage is labeled "Read From File" you would specify:

   ```
   Read\ From\ File=file:C:/inputfile.txt
   ```

   If the input stage is labeled "Illinois Customers" you would specify:

   ```
   Illinois\ Customers=file:C:/inputfile.txt
   ```

   When overriding a Read from File or Write to File location be sure to follow these guidelines:

   • Start the path with the "file:" protocol. For example, on Windows specify "`file:C:/myfile.txt`" and on Unix or Linux specify "`file:/testfiles/myfile.txt`".
   • The contents of the file must use an ASCII-based ISO-8559-1 (Latin-1) compatible character encoding.
   • You must use forward slashes (/) in file paths, not backslashes.
   • Spaces in stage names need to be escaped with a backslash.
   • Stage names are case sensitive.

      Note: If there are multiple schedules that use a control file trigger, it is important that they each monitor different control files. Otherwise, the same control file may trigger

multiple jobs or process flows causing unexpected behavior. For organizational purposes we recommend putting all required files and the control file in a dedicated directory.

10. In the **Poll interval** field, specify how frequently to check for the presence of the control file. For example, if you specify 10, the monitor will look every 10 seconds to see if the control file is in the monitored folder.

   The default is 60 seconds.

11. In the **Working folder** field, specify a folder where the control file will reside temporarily while the flow runs. Spectrum™ Technology Platform copies the file from the monitored folder to the working folder before running the flow. This clears out the monitored folder, which prevents the flow from being kicked off again by the same control file.

12. In the **Working folder options** field, specify what to do with the files in the working folder when the flow finishes running.

Keep	Leaves the files in their current location with their current name. If you select this option, the files in the working folder will be overwritten each time this schedule runs.
Move to	Moves the files from the working folder to a folder you specify. This allows you to preserve the files that were in the working folder by moving them to another location so that they are not overwritten the next time the file monitor runs. You can also use this option to move the files to another monitored folder to trigger a downstream process, like another dataflow or some other process.
Rename with time stamp	Adds a time stamp to the file name in the working folder. This allows you to preserve a copy of the files in the working folder since the renamed file will have a unique name and so will not be overwritten the next time the monitor runs a dataflow.
Delete	Deletes the files from the working folder after the flow finishes running.

13. If the flow is configured to send email notifications you can specify additional recipients for the notifications that will be sent when this schedule runs. The recipients you specify here will receive notifications in addition to those recipients specified in the flow's notification settings. To configure a flow to send notifications, open the flow in Enterprise Designer and go to **Edit** > **Notifications**.

14. Click **Save**.

Example: Monitored Folder and Working Folder

Let's say you have a car repair shop. Each day you want to mail the previous day's customers a coupon for a discount on future service. To accomplish this, you have a dataflow that takes the list of customers for the day, ensures the names have the correct casing, and validates the address. The list of customers for the day is generated by another system every evening. This other system generates a file containing the customer list, and you want to use this file as the input to the dataflow.

The system that generates the customer list puts it in a folder named `DailyCustomerReport`. It also places a blank trigger file in the folder when it is done. So you configure Spectrum™ Technology Platform to monitor this folder, specifying the following as the trigger file:

`C:\DailyCustomerReport\*.trg`

This tells Spectrum™ Technology Platform to run the dataflow whenever any file with a .trg extension appears in this folder. You could also specify a specific file name, but in this example we are using a wild card.

When a .trg file is detected in the `DailyCustomerReport` folder, Spectrum™ Technology Platform needs to move it to another folder before running the dataflow. The file must be moved because otherwise it would be detected again at the next polling interval, and this would result in the dataflow running again. So the file is moved to a "working folder" where it resides during the execution of the dataflow. You choose as the working folder `C:\SpectrumWorkingFolder`.

After the dataflow is finished processing the customer list, you want the trigger file to be moved to another location where it will trigger another process for billing. So, you select the **Move to** option and choose a folder named `C:\DailyBilling`.

So in this example, the trigger file starts off in `C:\DailyCustomerReport` and is then moved to the working folder `C:\SpectrumWorkingFolder`. After the dataflow is done, the trigger file is moved to `C:\DailyBilling` to initiate the billing process.

Viewing Flow Status and History

You can view a history of job, process flow, and service execution in Management Console and Enterprise Designer.

In Management Console

To view flow status and history in Management Console, go to **Flows** > **History**. The **Flows** tab shows job and process flow history, and the **Transactions** tab shows services history.

> **Note:** For flow history, the record counts shown when you hover over the **Results** column reflect the total number of records written as output by all the dataflow's sinks. This number may differ from the number of input records if the dataflow combines records, splits records, or creates new records.

By default, transaction history is disabled because enabling transaction history can have an adverse impact on performance. If you want to see transaction history you must turn on transaction history logging by clicking the **Transaction logging** switch. To view user activity, consider using the audit log which you can access under **System** > **Logs**.

The flow history list updates automatically every 30 seconds. If you want to update it sooner, click the Refresh button 🔄.

In Enterprise Designer

To view flow status and history in Enterprise Designer, go to **View** > **Execution History**.

The flow history list updates automatically every 30 seconds. If you experience slowness when viewing execution history uncheck the **Auto refresh** box.

The **Jobs** tab is used to monitor job status and to pause, resume, or cancel jobs that are running as well as delete completed jobs.

> **Note:** The record counts shown on the **Jobs** tab reflect the total number of records written as output by all the dataflow's sinks. This number may differ from the number of input records if the dataflow combines records, splits records, or creates new records.

- The **Succeeded** column shows the total number of records written as output by all the dataflow's sinks that have an empty value in the Status field.
- The **Failed** column shows the total number of records written as output by the dataflow's sinks that have a value of F in the Status field.
- The **Malformed** column shows the total number of records coming out of all source stage error ports.

The **Process Flows** tab is used to monitor process flow status and to cancel process flows that are running as well as delete completed process flows. If you click the plus sign next to any given process flow, you will view Activity Status information for the process flow. The following information is included in this area:

ActivityName	Includes the names of all activities, including any success activities, that make up the process flow.
State	The status of the activity (failed, succeeded, running, canceled).
ReturnCode	A code that indicates the result of the process flow. One of the following:

1	The process flow failed.
0	The process flow finished successfully.
-1	The process flow was canceled.
Other numbers	If the process flow contains a Run Program activity, the external program may return codes of its own. Any values in the ReturnCode column other than 1, 0, and -1 are from the external program. See the external program's documentation for an explanation of its return codes.

Started	The date and time the activity started.
Finished	The date and time the activity ended.
Comment	Any comments associated with the activity.

Downloading Flow History

You can download the information shown in the History page in Management Console to a Microsoft Excel file.

1. Open Management Console.
2. Go to **Flows** > **History**.
3. To download history information for services, click **Transaction History**. To download history for jobs and process flows, leave the **Flows** tab selected.
4. Click the Download button ⬇ .

    Tip: If you want to download only certain entries in the history list, modify the filter settings to show only the history you want to download.

Setting the Malformed Records Default

A malformed record is an input record that Spectrum™ Technology Platform cannot parse. By default, if the input data for a job contains one malformed record, the job will terminate. You can change this setting to allow more malformed input records, or even allow an unlimited number of malformed records. This procedure describes how to set a default malformed record threshold for jobs on your system.

> **Note:** You can override the default malformed record threshold for a job by opening the job in Enterprise Designer and going to **Edit** > **Job Options**.

1. Open Management Console.
2. Go to **Flows** > **Defaults**.
3. Click **Malformed Records**.
4. Select one of the following:

Terminate jobs containing this many malformed records	Select this option to terminate jobs if the input data contains one or more malformed records. Enter the number of malformed records that you want to trigger the termination of the job. The default is 1.
Do not terminate flows with malformed records	Select this option to allow an unlimited number of malformed records in the input data.

Setting Report Defaults

Reports are generated by jobs that contain a report stage. Reports can include processing summaries such as the number of records processed by the job, or postal forms such as the USPS CASS 3553 form. Some modules come with predefined reports. You can also create custom reports. Setting report defaults establishes the default settings for saving reports. The default settings can be overridden for a job, or for a particular report with in a job, by using Enterprise Designer.

This procedure describes how to set the default reporting options for your system.

1. Open Management Console.
2. Go to **Flows** > **Defaults**.
3. Click **Reports**.
4. Choose the format you want to use to save reports. Reports can be saved as HTML, PDF, or text.
5. Choose where you want to save reports.

Save reports to job history	Saves reports on the server as part of the job history. This makes it convenient for Management Console and Enterprise Designer users to view reports since the reports are available in the execution history.
Save reports to a file	Saves reports to a file in the location you specify. This is useful if you want to share reports with people who are not Spectrum™ Technology Platform users. It is also useful if you want to create an archive of reports in a different location. To view reports saved in this manner you can use any tool that can open the report's format, such as a PDF viewer for PDF reports or a web browser for HTML reports.

6. If you selected **Save reports to a file**, complete these fields.

Report location	The folder where you want to save reports.
Append to report name	Specifies variable information to include in the file name. You can choose one or more of the following:

Job ID	A unique ID assigned to a job execution. The first time you run a job on your system the job has an ID of 1. The second time you run a job, either the same job or another job, it has a job ID of 2, and so on.
Stage	The name of the stage that contributed data to the report, as specified in the report stage in Enterprise Designer.
Date	The day, month, and year that the report was created.

Overwrite existing reports	Replaces previous reports that have the same file name with the new report. If you do not select this option and there is an existing report that

has the same name as the new report, the job will complete successfully but the new report will not be saved. A comment will appear in the execution history indicating that the report was not saved.

Exposing a Service

Exposing a Service as a Web Service

Spectrum™ Technology Platform services can be made available as REST and/or SOAP web services. To make a service available on your server as a web service:

1. Open Enterprise Designer.
2. Open the service that you want to expose as a web service.
3. Go to **Edit** > **Web Service Options**.
4. To make the service available as a SOAP web service, check the box **Expose as SOAP web service**.
5. To make the service available as a REST web service, check the box **Expose as REST web service** and complete the following steps.

   a) If you want to override the default endpoint, specify the endpoint you want to use in the **Path** field.

   Specifying a path is optional. By default, a REST web service's endpoint is:

   `http://`*`server`*`:`*`port`*`/rest/`*`service_name`*`/results.`*`qualifier`*

   If you want to use a different endpoint, the path you specify is added after the service name. For example, if you specify `Americas/Shipping` in the **Path** field, your JSON endpoint would be something like this:

   `http://myserver:8080/rest/MyService/`**`Americas/Shipping`**`/results.json`

   You can use fields and options from the dataflow as variable names in the path by clicking the **Insert variable** drop-down menu and selecting the field or option you want to use. The variable is represented in the path using the notation `${Option.`*`Name`*`}` for dataflow options and `${Data.`*`Name`*`}` for dataflow fields.

   b) By default REST web services support the GET method and return data in XML and JSON formats. You can define additional HTTP methods and output formats by clicking **Add** to add a resource to the web service.

When you add a resource, you can choose the HTTP method (**GET** or **POST**). The supported data formats are listed below. You may not have all these formats available to you because some formats are only available if you have certain modules installed on your Spectrum™ Technology Platform server.

XML The default XML format. Use this format if you want to use XML as the format for requests and responses, and there is no specialized XML format for the kind of data you want to process.

JSON The default JSON format. Use this format if you want to use JSON as the format for requests and responses, and there is no specialized JSON format for the kind of data you want to process.

GeoJSON A specialized JSON format that is appropriate for services that handle geographic data. Support is provided only for Geometry and for the following the native platform types:

- boolean
- double
- float
- integer
- bigdecimal
- long
- string
- date
- time
- datetime
- timespan

If you try to expose a flow with any other type, you will not be able to specify GeoJSON (an error will appear at design-time). Also, GeoJSON only allows a single geometry. If the output contains multiple geometry fields, the system will search for a field called "geometry" followed by a field called "obj." If those fields do not exist, the first geometry field will be selected.

   c) Click **OK**.

   The new resource is added to the web service.

6. Click **OK** when you are done configuring the web service options.
7. Click the gray light bulb in the tool bar to expose the service.

When a dataflow is exposed the light bulb button in the Enterprise Designer tool bar indicates that the dataflow is exposed as shown here:

To verify that the service is now exposed as a web service, go to one of the following URLs:

- For REST: `http://server:port/rest`
- For SOAP: `http://server:port/soap`

Where *server* is the name or IP address of your Spectrum™ Technology Platform server and *port* is the port used for HTTP communication.

Exposing a Service for API Access

In order for a service to be accessible from through the Spectrum™ Technology Platform API, you must expose the service.

1. Open the service in Enterprise Designer.
2. Click the gray light bulb in the tool bar to expose the service.

    When a dataflow is exposed the light bulb button in the Enterprise Designer tool bar indicates that the dataflow is exposed as shown here:

Runtime Options

Adding Dataflow Runtime Options

Dataflow runtime options enable you control the behavior of stages when you run the dataflow. This is useful when you want to have the ability to modify the behavior of the dataflow when it runs. For example, you may want to specify a source database for a Read from DB stage when you run the dataflow, rather than using the database specified in the Read from DB stage in the dataflow.

This procedure describes how to expose options that can be set at runtime. After performing this procedure you will be able to set dataflow options at runtime using these techniques:

- For jobs, you will be able to specify runtime options using a dataflow options property file and job executor's `-o` argument.
- For services, you will be able to specify runtime options as API options.
- For services exposed as web service, you will be able to specify runtime options as parameters in the request.

- For subflows, runtime options will be inherited by the parent dataflow and exposed through one of the above means, depending on the parent dataflow type (job, service, or service exposed as a web service).

To add runtime options to a dataflow,

1. Open the dataflow in Enterprise Designer.
2. If you want to configure runtime options for a stage in an embedded dataflow, open the embedded dataflow.
3. Click the Dataflow Options icon on the toolbar or click **Edit** > **Dataflow Options**. The **Dataflow Options** dialog box appears.
4. Click **Add**. The **Define Dataflow Options** dialog box appears.
5. In the **Option name** field, specify the name you want to use for this option. This is the option name that will have to be specified at runtime in order to set this option.
6. In the **Label** field, you can specify a different label or keep it the same as the option name.
7. Enter a description of the option in the **Description** field.
8. In the **Target** field, chose whether you want this option to be applied to all stages in the dataflow or only certain stages.

    Selected stage(s)

    Select this option if you want the option to only be applied to the stages you specify.

    All stages

    Select this option if you want the option to be applied to all stages in the dataflow.

    Includes transforms

    Select this option if you want the runtime option to be made available to custom transforms in Transformer stages in the dataflow. If you choose this option you can access the value specified at run time in the Groovy script of a custom transform by using the following syntax:

    ```
    options.get("optionName")
    ```

    For example, to access an option named `casing`, you would include this in your custom transform script:

    ```
    options.get("casing")
    ```

9. If you chose **Selected stage(s)** in the **Target** field, the **Map dataflow options to stages** table displays a list of the stages in the dataflow. Select the option that you want to expose as a dataflow option. You will see the **Default value** and **Legal values** fields be completed with data when you select your first item.

    Note: You can select multiple options so that the dataflow option can control multiple stages options. If you do this, each of the stage options you select must share legal values. For example, one option has values of Y and N, each of the additional options must have either Y or N in their set of values, and you can only allow the value in common

to be available at runtime. So, if you select an option with Y and N values, you cannot select an option with the values of E, T, M, and L, but you could select an option with the values of P, S, and N because both options share "N" as a value. However, only "N" would be an available value for this option, not "Y", "P", or "S".

10. If you want to limit the values that can be specified at runtime, edit the options in the **Legal values** field by clicking on the icon just to the right of the field.

11. If you want to change the default value, specify a different value in the **Default value** field.

> **Note:** For a service, you can only modify default values before exposing the service for the first time. Once you expose the service you can no longer modify default values using Enterprise Designer. Instead, you must use Management Console. For more information, see **Specifying Default Service Options** on page 170.

12. Click **OK**.

13. Continue adding options as desired.

14. Click **OK** in the **Dataflow Options** dialog box when you are done adding options.

15. If you added a runtime option to an embedded dataflow, you must define the runtime option parent dataflow as well as all ancestor dataflows in order to make the options available at runtime. To do this, open the dataflow that contains the embedded dataflow and expose the option you just created. If necessary, open the parent of that dataflow and define the option there, and so on until all ancestors have the dataflow option defined.

For example, say you had a dataflow named "A" that contained an embedded dataflow named "B" which contained an embedded dataflow named "C", so that you have an embedded dataflow hierarchy like this: A > B > C. If you wanted to expose an option named Casing in a stage in embedded dataflow "C", you would open embedded dataflow "C" and define it. Then, you would open embedded dataflow "B" and define the option. Finally, you would open dataflow "A" and define the option, making it available at runtime.

The dataflow is now configured to allow options to be specified at runtime.

Specifying Default Service Options

Default service options control the default behavior of each service on your system. You can specify a default value for each option in a service. The default option setting takes effect when an API call or web service request does not explicitly define a value for a given option. Default service options are also the settings used by default when you create a dataflow in Enterprise Designer using this service.

> **Note:** For a service, you can only modify default values before exposing the service for the first time. Once you expose the service you can no longer modify default values using Enterprise Designer. Instead, you must use Management Console.

1. Open Management Console.
2. Click **Services**.
3. Check the box next to the service you want then click the Edit button [✎].
4. Set the options for the service. For information about the service's options, see the solution guide for the service's module.
5. Click **Save**.

Deleting Dataflow Runtime Options

Dataflow runtime options enable you to set stage options at runtime. The stage options can be set when calling the job through a process flow or through the job executor command-line tool. This procedure describes how to remove a dataflow's runtime option so that the option can no longer be set at runtime.

1. Open the job, service, or subflow.
2. Click the **Dataflow Options** icon or click **Edit** > **Dataflow Options**. The **Dataflow Options** dialog box appears.
3. Highlight the option you want to delete and click **Remove**.

Configuring Email Notification for a Dataflow

A dataflow can be configured to send an email notification containing job status information. For example you may want to send an email alert if a dataflow fails. The email notification can contain information such as the flow name, the start and end time, the number of records processed, and more.

> **Note:** A mail server must be configured in Management Console before you can set up notification for a dataflow. See the *Spectrum™ Technology Platform Administration Guide* for more information.

1. With a dataflow or process flow open in Enterprise Designer, select **Edit** > **Notifications**.
2. Click **Add**.
3. In the **Send Notification To** field, enter the e-mail address to which notifications should be sent.
4. Select the events you want to be notified about.
5. In the **Subject** field, enter the text you would like to appear in the subject line of the e-mail.
6. In the **Message** field, enter the text you would like to appear in the body of the e-mail.

You can choose to include information about the job in the email by clicking **Click Here to Insert a Field in the Subject or Message**. Some examples of job information are: start time, end time, and number of records failed.

7. Click **Preview** if you wish to see what the notification will look like.
8. Click **OK**. The **Notifications** dialog box will reappear with the new notification listed.
9. Click **OK**.

5 - Combining Flows into a Process Flow

In this section

Introduction to Process Flows

A process flow executes a series of activities such as jobs and external applications. Each activity in the process flow executes after the previous activity finishes. Process flows are useful if you want to run multiple dataflows in sequence or if you want to run an external program. For example, a process flow could run a job to standardize names, validate addresses, then invoke an external application to sort the records into the proper sequence to claim postal discounts. Such a process flow would look like this:

In this example, the jobs Standardize Names and Validate Addresses are exposed jobs on the Spectrum™ Technology Platform server. Run Program invokes an external application, and the Success activity indicates the end of the process flow.

Designing Process Flows

Creating a Process Flow

To create a process flow use Enterprise Designer to create a sequence of activities that execute jobs and/or external applications.

1. Open Enterprise Designer.
2. Select **File** > **New** > **Process Flow**.
3. Add the first action you want the process flow to perform. You can do one of the following:

    • To execute a job, drag the job's icon from the **Activities** folder in the palette to the canvas.
    • To execute an external program, drag a Run Program icon from the **Activities** folder in the palette to the canvas.

4. Add the second action you want the process flow to perform.

    You can add a job by dragging a job's icon to the canvas, or add an external program by dragging a Run Program icon onto the canvas.

5. Connect the two activities by clicking the gray triangle on the right side of the first icon and dragging it to connect to the gray icon on the left side of the second icon.

   For example, if you have a process flow that first executes a job named GeocodeAddress then runs an external program, your process flow would look like this:

6. Add additional activities as needed.

7. When you have added all the activities you want to execute in the process flow, drag a Success activity onto the canvas and connect it to the last activity in the process flow.

   For example, this process flow contains two jobs ("Standardize Names" and "Validate Addresses") and one Run Program activity. At the end of this process flow is the Success activity:

8. Double-click the activities you placed on the canvas to configure their runtime options. You can also double-click the connection between activities to configure transition options.

Using a Variable to Reference a File

In a process flow, variables are useful if you want multiple activities in the process flow to reference the same file. Using a variable, can define the file in one place, then reference the variable in all downstream activities that need to reference the file. If the file ever changes, you can modify the variable definition without having to modify all the downstream activities.

When you add a job activity to a process flow, the activity automatically creates variables for each source and sink in the dataflow. If there are files you want to use in the process flow that are not defined in the source or sink of a job, you can create variables.

When you add a Run Program activity, no variables are created by default. If you want to use variables with a Run Program activity you must create them.

This procedure describes how to create a variable in a job activity or a Run Program activity.

1. Open the process flow in Enterprise Designer.

2. Double-click the job activity or Run Program activity where you want to define the variable.

   Note: Variables can only be referenced by activities that follow the activity where you define the variable, so be sure to define the variable in an activity that precedes the activities where you want to use the variable.

3. Click the **Variables** tab.
4. Create the variable.

Option	Description
To create a new variable for an input file...	Next to **Inputs** click **Add**. In the **Name** field enter a name for the variable. This is the name that downstream activities will reference. In the **Location** field choose one of the following:

	Use file specified in job	Choose this option to use the file defined in the source stage in the job. This option is only available if you are defining a variable for a job activity.
	Browse for file on the server	Choose this option if you want to select a file to assign to this variable.
	Reference an upstream activity's file	Choose this option if you want to use a file assigned to an existing variable from an upstream stage.

To create a new variable for an output file...	Next to **Inputs** click **Add**. In the **Name** field enter a name for the variable. This is the name that downstream activities will reference. In the **Location** field choose one of the following:

	Browse for file on the server	Choose this option if you want to select a file to assign to this variable.
	Temporary file managed by the server	Choose this option if you want this variable to reference a temporary file that will be automatically created and deleted as needed. This option is useful in cases where a file used only as an intermediate step in a process flow and is not needed once the process flow completes.

5. Click **OK** to close the **Add Variable** window.
6. Click **OK** to close the activity options window.
7. To reference the variable in a downstream activity:
   a) Double-click the activity that you want to reference the variable.
   b) Select the input stage that you want to have reference the variable and click **Modify**.
   c) In the **Location** field, select Reference an upstream activity's file....

Adding Conditional Logic to a Process Flow

You can add conditional logic to a process flow so that different activities are executed based on the return code of a preceding activity. For example, you could execute one activity if a job returns a return code of 1 and another activity if a job returns a return code of 0. In this way you can build conditional branching into your process flow.

1. Open the process flow in Enterprise Designer.
2. Double-click the transition between two activities of the flow.

   A *transition* is the line that connects two activities. For example, the line between the GeocodeAddress activity and the Run Program activity shown here is a transition:

   The **Transition Options** window appears.

3. Select the type of transition you wish to add.

Simple	Select this option if you always want this path in the process flow to execute.
Conditional	Select this option if you only want this path in the process flow to execute if the upstream activity returns a specific return code or return codes, or a range of return codes.
Otherwise	Select this option if you want this path in the process flow to execute only if the conditions in the other transitions leading from the activity are not met.

   > **Note:** Only one **Otherwise** transition can exist among the transitions leading from an activity.

4. Click **OK**.
5. To configure which transitions trigger an activity, right-click the activity, select **Input modes**, then choose one of the following:

First Input	The first transition coming into this activity, whether through a **Simple**, **Conditional**, or **Otherwise** transition, triggers the execution of the activity. Other transitions are ignored.

All Inputs	The activity does not run unless all transitions coming into this activity are taken.

6. To configure which transitions leading out of an activity are taken, right-click the activity, select **Output modes**, then choose one of the following:

First Output	The first transition that evaluates to true is taken. Other transitions are ignored, even if their conditions evaluate to true.
All Outputs	All transitions that evaluate to true are taken.

Deleting a Process Flow

1. Go to **File** > **Manage**. The **Manage** dialog box will appear.
2. Right-click on the process flow you want to delete and select **Delete**.
3. Click **OK**.

Activities

Job

A job activity executes a job as part of a process flow. The following example shows a process flow that executes two job activities: Standardize Names and Validate Addresses.

To add a job activity to a process flow, drag the job activity from the Activities folder in the palette to the canvas.

> **Note:** In order for a job to be available to use in a process flow, the job must be exposed. If the job you want is not exposed, open the job in Enterprise Designer and select **File** > **Expose**.

Double-click the job activity to configure the **Options** tab and the **Variables** tab.

Options Tab

The Options tab displays the runtime options available for the job. You can change the runtime options so that when the job is executed in this process flow the options you specify here are used for the job. For example, if one of the job's dataflow options is controls the units to use for distance

and defaults to miles, you could override that option here and have distance returned in kilometers instead when the job is executed through this process flow.

Variables Tab

The Variables tab displays the source and sink stages in the dataflow. You can override the input and output files specified in the dataflow so that when the job is executed in this process flow, different input and/or output files are used.

Overriding Input and Output Files

By default, a job executed in a process flow uses the input and output files defined in the job's source and sink stages. You can, however, override the input and output files defined in the job so that when the process flow is executed, the job will use the input or output file you specify in the job activity in the process flow instead of the file specified in the job's source or sink stages. You can override input and output files by specifying a specific file or by using a variable to refer to a file defined in an upstream activity.

1. Open the process flow in Enterprise Designer.
2. Double-click the job activity for which you want to override an input or output file.
3. Click the **Variables** tab.

    On the **Variables** tab, the variables listed under **Inputs** correspond to the source stages in the job. The variables listed under **Outputs** correspond to the sink stages in the job.

    For example, say you have a process flow that contains a job activity for a job named MyGeocodingJob. On the **Variables** tab of the activity options you see this:

Each variable listed corresponds to the name of a source or sink stage in the MyGeocodingJob dataflow. In this example the **Variables** tab shows one source (Read from File) and two sinks (Write Failed and Write Successful). If you were to open up the MyGeocodingJob dataflow in Enterprise Designer, you would see something like this:

4. On the **Variables** tab, select the source stage or output stage that you want to override and click **Modify**.

5. Do one of the following:

Option	Description
To override an input file...	In the **Location** field choose one of the following:

Option		Description
	Use file specified in job	Choose this option to use the file defined in the source stage in the job.
	Browse for file on the server	Choose this option if you want to override the file defined in the job and use a different file that you choose.
	Reference an upstream activity's file	Choose this option if you want to override the file defined in the job with a file whose name and location is defined in an upstream activity's Read from File or Write to File stage or an upstream activity's variable. Use this option if the output file from a previous activity is the input for this activity. The advantage of this option is that if the upstream activity's Write to File stage is ever modified to point to another file, this activity will still point to the correct file. Another advantage is that you do not need to know the file path and name of upstream activities' input file to point to it.
To override an output file...	In the **Location** field choose one of the following:	
	Browse for file on the server	Choose this option if you want to override the file defined in the job and use a different file that you choose.
	Temporary file managed by the server	Choose this option if you want this variable to reference a temporary file that will be automatically created and deleted as needed. This option is useful in cases where a file used only as an intermediate step in a process flow and is not needed once the process flow completes.

6. Click **OK** to close the **Modify Variable** window.
7. Click **OK** to close the activity options window.

When the process flow runs, the job will use the files you specified in the process flow activity instead of the file specified in the job itself.

Clear Cache

A Clear Cache activity clears the global cache data as a part of process flow. It does not delete the cache but only clears the cache data.

1. Drag the **Clear Cache** activity to the canvas.
2. Double click the Clear Cache.

3. Select the cache. You can also select multiple caches to clear their data.
4. Run the process flow.

Execute SQL

An Execute SQL activity allows you to run the SQL statements both before and after the execution of dataflow or an external program.

Load to Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. To use Hive to query the underlying data source, use its own query language, HiveQL.

Hive supports the below Hadoop file formats:

- TEXTFILE
- SEQUENCE FILE
- ORC
- RCFILE
- PARQUET
- AVRO

> **Note:** The AVRO file format is supported in Hive version 0.14 and higher.

The **Load to Hive** activity allows you to load data into a Hive table using a JDBC connection. Using this connection, data is read from a specified Hadoop file and loaded to either an existing table of a selected connection, or to a newly created table in the selected connection.

To load the data to a new table, the schema of the table needs to be defined.

> **Note:** Spectrum does not support hierarchical data, even though Hive supports it.

Creating a Hive Connection

1. Open the **Load to Hive** activity.
2. From the **File name** field, enter the name of the file, which is to be read. Click Browse [...] to select the file to be read.
3. From the **File type** field, select the format of the file to be read. The default file format selected is `Delimited`.

    If the **File type** selected is either `Delimited` or `Sequence`, the fields **Field Separator** and **Record Separator** are displayed. Else, they are not displayed.
4. In the **Field Separator** field, select the character that separates each consecutive field of a record.
5. Select the connection for the Hive database you want to use in the **Connection** field.

    a) To add, modify, and delete connections, click **Manage**.
    The **Database Connection Manager** window opens.

    b) Click **Add** to create a new connection, or **Modify** to edit an existing connection.
    The **Connection Properties** window opens.

    c) Enter the **Connection Name**.

    d) In the **Database Driver** field, select a Hive database driver for the connection.

    e) Specify all the details of the connection, namely **user**, **password**, **host**, **port**, and **instance**.

    f) To test the connection details, click **Test**.

    g) If the connection test is successful, click **OK**.
    The **Connection Properties** window closes.

    h) Click **OK**.
    The **Database Connection Manager** window closes.

6. In the **Table/View** field, select the table you wish to write to, or type in the name of a new table to be created.

   If you create a new table in the **Table/View** field, the **External** checkbox gets enabled. Else, if you select an existing table, the **External** checkbox remains disabled.

7. To create the new table external to the Hive database, check the **External** checkbox.

   Important: In case of External tables:

   1. Existing records cannot be overwritten, and new records cannot be added. You are only allowed to create new external tables and populate them with records.

   2. If you select a file placed in a particular folder, all files placed in that folder are automatically selected. Hence, ensure all files placed in the particular folder have the same format.

   Learn more about Hive EXTERNAL tables **here**.

8. To overwrite all existing records of the table, check the **Overwrite** checkbox. This deletes existing records of the selected table, and adds the records read from the file to the table.

9. The grid displays the names and datatypes of the columns of the selected table.

   If you have specified a new table in the **Table/View** field, use the **Add**, **Modify** and **Remove** buttons beside the grid to add columns to define the table, and specify their respective datatypes. Use the **Move Up** and **Move Down** buttons to specify the sequence of the table columns.

   > **Note:** The **Add**, **Modify**, **Remove**, **Move Up**, and **Move Down** buttons remain disabled if you select an existing table in the **Table/View** field.

   Important:

   1. Ensure the datatypes of all the fields in the file match the datatypes of the respective table columns, unless all datatypes are of String type. Else, the data load may result in inconsistent data.

   2. Ensure the number of fields in the file match the number of table columns. Else, the data in the extra fields in the file are discarded.

3. Hive accepts names of tables and columns in small case only. If you enter the names using block letters, Hive converts them to small case. The resultant schema displays all names in small case.

10. Click **OK**.

> **Note:** If you opt to create a new table and define its columns, the same is created at runtime. The **Load to Hive** activity is only to design the table structure. At runtime, the designed table is created and the data read from the file is written into it.

Run Hadoop MapReduce Job

The **Run Hadoop MapReduce Job** activity allows you to run any MapReduce job on a Hadoop cluster, by mapping the relevant JAR file. You can use this activity to run a MapReduce job of the Spectrum™ Data Quality for Big Data SDK or any external MapReduce job.

> **Note:** If the MapReduce job fails, an error message is displayed along with the status of the job run.

For a list of supported job configurations on both Windows and Linux platforms, see the table **Supported job configurations** on page 189.

Field	Description
Hadoop server	The list of configured Hadoop servers.
	For information about mapping HDFS file servers through Management Console, see the *Administration Guide*.
Jar path	The path of the relevant JAR file for the Hadoop MapReduce job to run.
	Note: The JAR must be present at the external client location or Spectrum server. It must not be placed on the Hadoop cluster.
Driver class	Select one of:
	Default — To run an external job by simply entering the job's *class name* and *arguments* for the job, select `Default`.
	On selecting `Default`, the fields **Class name** and **Arguments** are displayed.
	Configure — To enter additional job properties of any external job or to run any one of the Spectrum Big Data Quality SDK jobs, select `Configure`.
	On selecting `Configure`, the field **Job type** is displayed.

Field	Description
Job type	Select one of:

	Spectrum	To run any one of the Spectrum Big Data Quality SDK jobs, select `Spectrum`.
		On selecting `Spectrum`, the field **Spectrum jobs** is displayed.
	Generic	To specify additional properties for any external job, select `Generic`.

Spectrum jobs	Select the required job from the list of Spectrum Big Data Quality SDK jobs. The list includes these jobs:

- Address Validation
- Advanced Transformer
- Best of Breed
- Duplicate Syncronization
- Filter
- Groovy
- Intraflow Match
- Interflow Match
- Joiner
- Match Key Generator
- Open Name Parser
- Open Parser
- Table Lookup
- Transactional Match
- Validate Address
- Validate Address Global

On selecting the desired Spectrum job:

1. The fields **Job name**, **Class name**, and **Arguments** are auto-populated.

   All the auto-populated fields can be edited as required, except **Class name**.

   Important: For the selected Spectrum job, the auto-populated **Class name** must not be edited, else the job cannot run.

2. The **Properties** grid is auto-populated with the required configuration properties of the selected Spectrum job, with their default values.

   You can add or import more properties as well as modify the auto-populated properties, as required.

Class name	The fully qualified name of the driver class of the job.

Field	Description

Arguments	The space-separated list of arguments. These are passed to the driver class at runtime to run the job.
	
	For example,
	
	`23Dec2016 /home/Hadoop/EYInc.txt`
	
	1. Those variables can be passed as arguments in the argument list, which are defined to accept runtime values either in the source stage or this current stage of the process flow.
	
	   For example, if in the output of the previous stage of the process flow the variable `SalesStartRange` is defined, you can include this variable in this argument list as `${SalesStartRange}` along with other required arguments, as illustrated:
	
	`23Dec2016 /home/Hadoop/EYInc.txt`
	`${SalesStartRange}`
	
	2. In case a particular argument contains a space, enclose it in double quotes.
	
	   For example, `"/home/Hadoop/Sales Records"`.
	
	Spectrum Big Data Quality SDK Jobs - Arguments:
	
	To run the Spectrum Big Data Quality SDK *MapReduce* jobs, pass the various configuration files as a list of arguments. Each argument key accepts the path of a single configuration property file, where each file contains multiple configuration properties.
	
	The syntax of the argument list for configuration properties is:
	
	`[-config <Path to configuration file>] [-debug] [-input <Path to input configuration file>] [-conf <Path to MapReduce configuration file>] [-output <Path of output directory>]`
	
	For example, for a MapReduce MatchKeyGenerator job:
	
	`-config /home/hadoop/matchkey/mkgConfig.xml -input /home/hadoop/matchkey/inputFileConfig.xml -conf /home/hadoop/matchkey/mapReduceConfig.xml -output /home/hadoop/matchkey/outputFileConfig.xml`
	
	   Note: If the same configuration property key is specified both in the **Arguments** field and in the **Properties** grid but each points to different configuration files, the file indicated in the **Properties** grid for this property holds.
	
	The sample configuration properties are shipped with the Data and Address Quality for Big Data SDK and are placed at the location `<Big Data Quality bundle>\samples\configuration`.

General Tab

Field	Description	Requirement
Job name	The name of the Hadoop MapReduce job.	Required
Input path	The path of the input file for the job.	Required
Output path	The path of the output file for the job.	Required
Overwrite output	Indicates if the specified output path must be overwritten in case it already exists. **Note:** If this check box is left unchecked, and the configured output path is found to exist at runtime, Hadoop throws an exception and the process flow is aborted.	Optional
Mapper class	The fully qualified name of the class that handles the Mapper functionality for the job.	Required
Reducer class	The fully qualified name of the class that handles the Reducer functionality for the job.	Optional
Combiner class	The fully qualified name of the class that handles the Combiner functionality for the job.	Optional
Partitioner class	The fully qualified name of the class that handles the Partitioner functionality for the job.	Optional
Number of reducers	The number of reducers used to run the MapReduce job.	Optional
Input format	The format of the input data.	Required
Output format	The format of the output data.	Required
Output key class	The datatype of the keys in the output key-value pairs.	Required
Output value class	The datatype of the values in the output key-value pairs.	Required

Properties Tab

To specify additional properties to run the required job, use this tab to define as many property-value pairs as required. You can add the required properties directly in the grid one at a time.

Alternatively, to import properties from a file, click **Import**. Go to the location of respective property file and select the file of XML format. The properties contained in the imported file are copied into the grid. The property file must be in XML format and must follow the syntax:

```
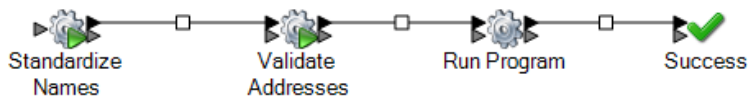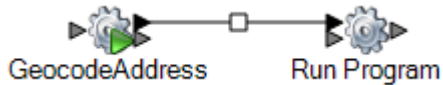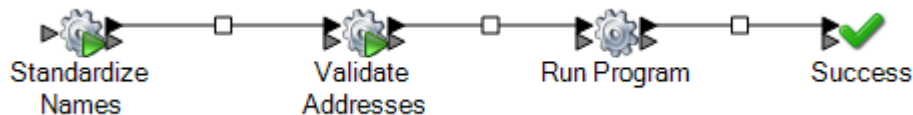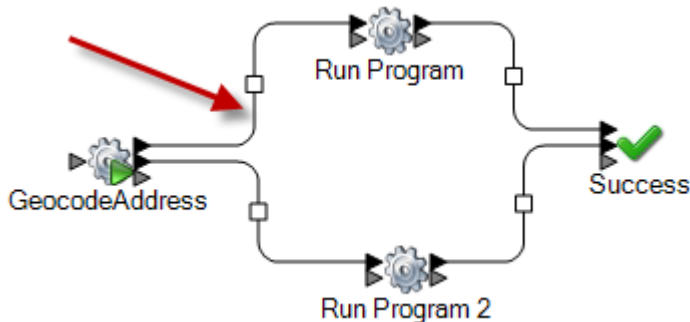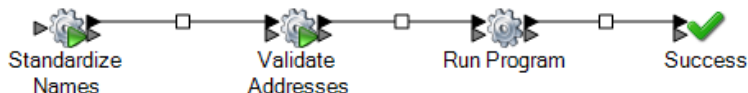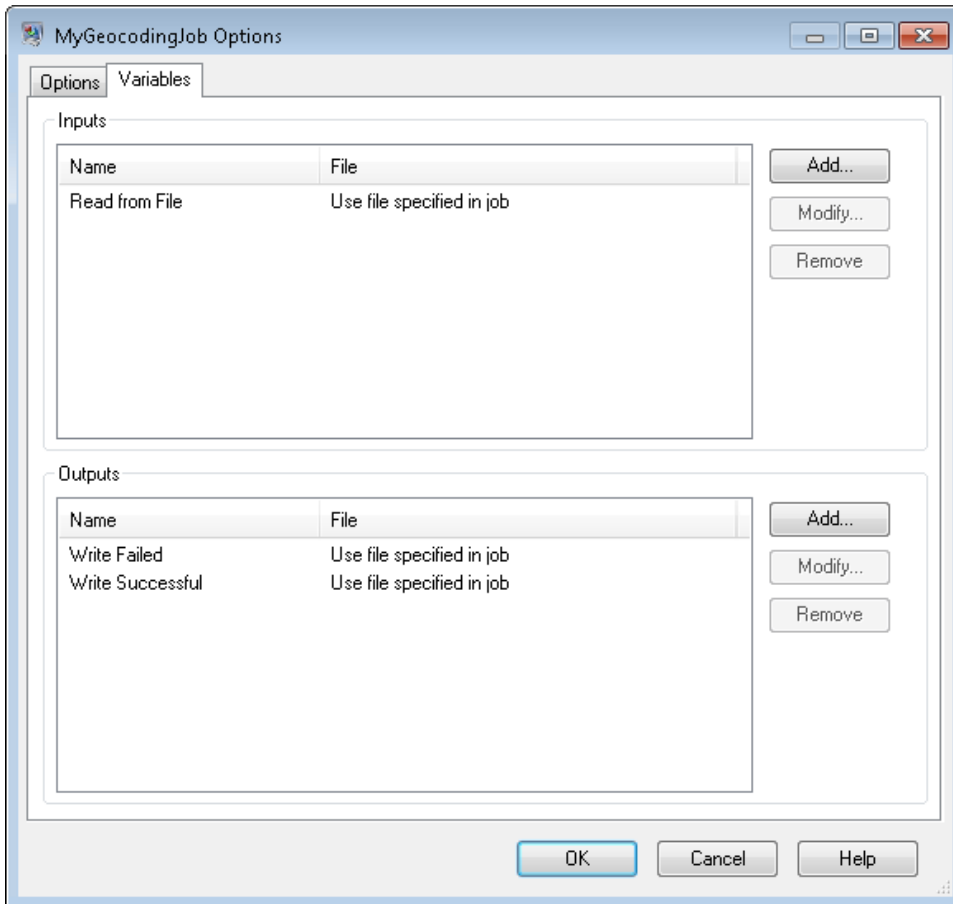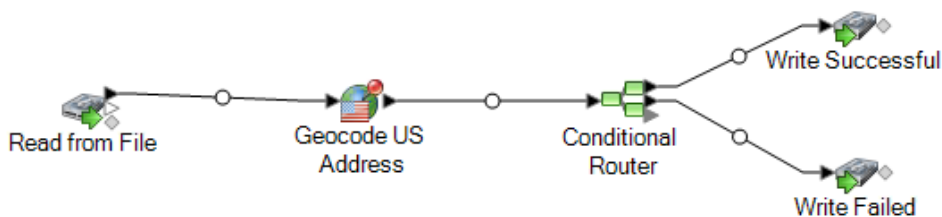<configuration>
    <property>
        <name>key</name>
        <value>some_value</value>
        <description>A brief description of the
            purpose of the property key.</description>
    </property>
</configuration>
```

You can directly import the Hadoop property file *mapred.xml*, or create your own files using this XML format.

> **Note:**
>
> 1. If the same property is defined here and in Management Console, the values defined here override the ones defined in Management Console.
> 2. If the same property exists both in the grid and also in the imported property file, then the value imported from the file overwrites the value existing in the grid for the same property.
> 3. You can import multiple property files one after the other, if required. The properties included in each imported file are added in the grid.
> 4. Ensure the property file is present on the Spectrum™ Technology Platform server itself.
> 5. The `<description>` tag is optional for each property key in a configuration property file.
> 6. Reference data needs to be placed local to data nodes to run the relevant jobs. This property is available only for jobs that use reference data, such as *Advanced Transformer*, *Validate Address Global*, and *Validate Address*. The property is: *pb.bdq.reference.data.location*.

Dependencies Tab

In this tab, add the list of the input files and Jar files required to run the job.

Once the job is run, the reference files and the reference Jar files added here are available in the distributed cache of the job.

Reference Files

> To add the various files required as input to run the job, click **Add**, go to the respective location on your local system or cluster, and select the particular file.
>
> To remove any file added in the list, select the particular file and click **Remove**.

Reference Jars

> To add the Jar files required to run the job, click **Add**, go to the respective location on your local system or cluster, and select the particular Jar file.
>
> To remove any file added in the list, select the particular file and click **Remove**.

Supported job configurations

This table lists the supported jobs in default and configured mode on Windows and Linux platforms:

	Windows		Linux	
Stages	Default mode	Configure mode	Default mode	Configure mode
MatchKeyGenerator	Supported	Supported	Supported	Supported
Intraflow Match	Supported	Supported	Supported	Supported
Interflow Match	Supported	Supported	Supported	Supported
Transactional Match	Supported	Supported	Supported	Supported
Table Lookup	Supported	Supported	Supported	Supported
Advanced Transformer	Supported	Supported	Supported	Supported
Open Name Parser	Supported	Supported	Supported	Supported
Duplicate Synchronization	Supported	Supported	Supported	Supported
Filter	Supported	Supported	Supported	Supported
Best of Breed	Supported	Supported	Supported	Supported
Open Parser	Supported	Supported	Supported	Supported
Address Validation (GAV)				
Groovy	Supported	Supported	Supported	Supported
Joiner	Supported	Supported	Supported	Supported
Validate Address Global (AD)	Supported	Supported	Supported	Supported
Validate Address (C1P)			Supported	Supported

Note:

- GAV and C1P stages are not supported on Windows platform where spectrum installation directory contains spaces.
- GAV and C1P stages behavior need to verify again on windows platform where spectrum installation directory does not contain any spaces.
- Reference data on HDFS strategy is not supported.

Variables

Inputs

In this grid, from among the fields received from a source activity, select the fields to be used in this activity.

Outputs

In this grid, select the fields to be included in the output to the destination activity in this process flow.

Run Hadoop Pig

Run Hadoop Pig runs an Apache Pig script. Apache Pig is a high-level language for expressing data analysis programs, and has the infrastructure for evaluating these programs. Pig programs can be parallelized and this enables them to handle very large data sets.

Run Hadoop Pig allows you to select the Pig operations,enter any needed parameters and have your Pig script automatically generated by the system. You can run the Pig script on any Hadoop server.

Run Hadoop Pig works only on Hadoop File Servers. Both Apache Hadoop *1.x* and *2.x* are supported.

To set the **Run Hadoop Pig** options:

1. Drag and drop the **Run Hadoop Pig** activity to the canvas.
2. Right-click the **Run Hadoop Pig** activity and select **Options**.
3. The server name fields indicates the Hadoop Server on which the file to be processed resides.
4. Click the browse button **([...])** to go to the file to be processed.
5. Select the file type. Run Hadoop Pig supports both delimited as well as delimited sequence files.
6. Select the delimiter and the text qualifier as appropriate.
7. Click **Add** from the Fields section and add the fields that are present in the file to be processed. For sequence files, the first field is considered the key and the other fields are part of the delimited values.
8. Select the **Trim** operation as desired. The trim operation trims white spaces in the input field, before processing it.

9. Go to the operations tab. Click **Add** to start adding the Pig operations to be performed on the file. This opens the Operations editor.

10. Select an operation to be performed. The various operations are as follows:

    • Sort - Sorts the data in alphabetical order.
    • Filter -Filters data according to your requirements.
    • Aggregate - Performs statistical operations, such as Sum and Count on the data.
    • Distinct - Selects all unique records from the specified field.
    • Limit - Limits the number of records processed to a specified number.

11. Use the **Move Up** and **Move Down** buttons to change the order of operations.

12. Once you have selected the operations and entered the required input for processing the operations, click **Add** to save your selection and return to the Pig options editor.

13. The Pig Script is automatically generated based on the selected operations.

    • The editor allows you to override the generated Pig script with your own script, as needed. Click the **Edit Script** option and enter your own script in the Pig Script text box. The **Regenerate** button is enabled in this case. If you want the system generated script again, click **Regenerate** from the Pig Script section to generate the Pig script.
    • To validate script syntax, click the **Validate** button.

14. You can specify the output file under **Variables** tab. The output file can be used by the subsequent activities.

15. Click **OK** to save the Pig Script. By default, the output file type is the same as the input file type. You can change this using the generated Pig script.

Hadoop Pig Operations

The various Pig operations are as follows:

1. Sort: Sorts the data in alphabetical order. The sort operation is described in detail in **Sorting Input Records**.

2. Filter: Allows you to filter the data according to your requirements. The filter operation is described in more detail in **Filtering Input Records**.

3. Aggregate: Allows you to perform statistical operations such as Sum, Count and others, on the data.

    Select the aggregate operations for each field as desired.

    • Sum: Calculates the sum of the values in the field.
    • Average: Calculates the average from all the values in the field.
    • Max: Calculates the maximum value from the values in the field.
    • Min: Calculates the minimum value from the values in the field.
    • Count: Calculates the total number of values in the field.

        Note: If you select the Distinct operation, only the values that are unique are counted.

4. Distinct: Selecting this option, causes the Aggregate Count operation to count only unique values in the field.

5. Limit: Enter a value greater than zero, to limit the number of records processed, to this value.

Run Program

The Run Program activity executes an external application as part of a process flow.

Table 9: Run Program Options

Option Name	Description
Program name	The path to the executable you wish to run.
Arguments	Specifies command line arguments to pass to the program specified in the **Program name** field. Separate multiple arguments with spaces. You can use variables defined on the Variables tab as arguments by clicking **Insert Variable**. For more information on variables, see **Using a Variable to Reference a File** on page 175.
Time out (in seconds)	Specifies an amount of time to wait for the program specified in the **Program name** field to respond. If the program is unresponsive for the amount of time specified the process flow will fail.
Environment variables	Specifies environment variable values to use when running this program. If you specify values here the program will use these environment variables instead of those specified on your system. Otherwise, it will use the environment variables specified on your system. Note that if the program you are calling uses multiple environment variables you must either define values for all of them or none of them. Specifying values here does not change the environment variable definitions on your system.
	Click **Add** and enter the name of the variable in the **Variable Name** field. An example might be "JAVA_HOME". Enter the value of the variable in the **Variable Value** field. An example might be "C:\Program Files\Java\jdk1.6.0_17." Instead of entering a value you can click **Insert Variable** to set it to the value of a variable defined in on the **Variables** tab.

Specifying Input and Output Files

You can call an external application from a process flow by using a Run Program activity. You can specify the file that contains the data you want to send to the external application as well as the output file you want the external application to write to.

1. In a process flow, double-click a Run Program activity.
2. Click the **Variables** tab.
3. To specify an input file,
   a) Click the **Add** button under the **Inputs** section.
   b) In the **Name** field, enter a meaningful name for this file. The name can be anything you choose.
   c) In the **Location** field choose one of the following:

Browse for file on the server	Choose this option if you want to browse to the input file you want and select it.
Reference an upstream activity's file	Choose this option if you want to use a file assigned to an existing variable from an upstream stage.

   d) Click **OK**.
4. To specify an output file,
   a) Click the **Add** button under the **Outputs** section.
   b) In the **Name** field, enter a meaningful name for this file. The name can be anything you choose.
   c) In the **Location** field choose one of the following:

Browse for file on the server	Choose this option if you want to browse to the output file you want and select it.
Temporary file managed by the server	Choose this option if you want the output from the program to be written to a temporary file that will be automatically created and deleted as needed. This option is useful in cases where a file used only as an intermediate step in a process flow and is not needed once the process flow completes.

   d) Click **OK**.
5. Click **OK** to close the **Run Program Options** window.

Using a Control File with an External Program

You can call an external application from a process flow by using a Run Program activity. In doing so you can use a control file that contains configuration settings for the external application. For example, you could call VeriMove™ with a Run Program activity and specify a control file to use during execution.

1. In a process flow, double-click a Run Program activity.

2. Click the **Variables** tab.

3. In the **Control Files** section, click **Add**.

4. In the **Name** field, give the control file a name. The name can be anything you choose.

5. In the **Contents** field, specify the contents of the control file.

   To reference a file using a variable defined under **Inputs** or **Outputs**, click **Insert Variable**.

6. Click **OK** to close the **Add Control File** window.

7. Click the **Options** tab.

8. Click **Insert Variable**.

9. Select the name of the control file you created then click **OK**.

   A variable that points to your control file is added to the **Arguments** field.

10. If necessary modify the **Arguments** field to use the necessary command line arguments to indicate a control file. See the documentation of the external application for more information.

11. Click **OK** to close the **Run Program Options** window.

Spark Sorter

The **Spark Sorter** activity allows you to sort a massive bulk of records. This activity uses Apache Spark libraries to power the feature and runs on your Spectrum™ Technology Platform server.

Currently, *delimited* files, present on the Spectrum™ Technology Platform server, are accepted to read the input records.

> **Note:** Files present on remote servers are not supported.

Field	Description
Server name	Indicates the location of the file you select as input.
	Since the **Spark Sorter** activity only accepts files located on the Spectrum™ Technology Platform, this field displays Spectrum™ Technology Platform.
File name	Specifies the path to the file. Click the ellipses button (...) to go to the file you want.
	You can read multiple files by using a wild card character to read data from multiple files in the directory. The wild card characters * and ? are supported. For example, you could specify `*.csv` to read in all files with a `.csv` extension located in the directory. In order to successfully read multiple files, each file must have the same layout (the same fields in the same positions). Any record that does not match the layout specified on the **Fields** tab will be treated as a malformed record.
	Attention: If the Spectrum™ Technology Platform server is running on Unix or Linux, remember that file names and paths on these platforms are case sensitive.

Field	Description
Record Type	The format of the records in the file. Currently, *delimited* file formats are accepted as input.

	Delimited	A text file in which records are separated by an end-of-line (EOL) character such as a carriage return or line feed (CR or LF) and each field is separated by a designated character such as a comma.

Character Encoding	The character encoding of the input file.

The encoding `UTF-8` is supported. For more information about UTF, see **unicode.org/faq/utf_bom.html**.

Field separator	Specifies the character used to separate fields in a delimited file.

For example, this record uses a pipe (|) as a field separator:

```
7200 13TH ST|MIAMI|FL|33144
```

These characters available to define as field separators are:

- Space
- Tab
- Comma
- Period
- Semicolon
- Pipe

If the file uses a different character as a field separator, click the ellipses button to select another character as a delimiter.

Text qualifier	The character used to surround text values in a delimited file.

For example, this record uses double quotes (") as a text qualifier.

```
"7200 13TH ST"|"MIAMI"|"FL"|"33144"
```

The characters available to define as text qualifiers are:

- Single quote (')
- Double quote (")

If the file uses a different text qualifier, click the ellipses button to select another character as a text qualifier.

Field	Description
Record separator	Specifies the character used to separate records in line a sequential or delimited file. This field is not available if you check the **Use default EOL** check box.

The record separator settings available are:

Unix (U+000A)	A line feed character separates the records. This is the standard record separator for Unix systems.
Macintosh (U+000D)	A carriage return character separates the records. This is the standard record separator for Macintosh systems.
Windows (U+000D U+000A)	A carriage return followed by a line feed separates the records. This is the standard record separator for Windows systems.

If your file uses a different record separator, click the ellipses button to select another character as a record separator.

Field	Description
Use default EOL	Specifies that the file's record separator is the default end of line (EOL) character used on the operating system on which the Spectrum™ Technology Platform server is running.

Do not select this option if the file uses an EOL character that is different from the default EOL character used on the server's operating system. For example, if the file uses a Windows EOL but the server is running on Linux, do not check this option. Instead, select the Windows option in the **Record separator** field.

Field	Description
First row is header record	Specifies whether the first record in a delimited file contains header information and not data.

For example, this file snippet shows a header row in the first record.

```
"AddressLine1"|"City"|"StateProvince"|"PostalCode"
"7200 13TH ST"|"MIAMI"|"FL"|"33144"
"One Global View"|"Troy"|"NY"|12180
```

Field	Description
Output	Specifies the path to the output file on the Spectrum™ Technology Platform server. Click the ellipses button (...) to go to the output directory and file name you want.

Attention: If the Spectrum™ Technology Platform server is running on Unix or Linux, remember that file names and paths on these platforms are case sensitive.

Field	Description
Overwrite	Indicates that the output file must overwrite if a file exists with the same name as specified in the **Output** field.
Concatenate	Indicates that all Spark part files must be concatenated into a single output file in the specified **Output** location.

Field	Description
Preview	Once the input file is selected in the **File name** field, the **Preview** grid displays the first 100 records of the existing output file.
	To correctly display all the separate column values, click **Regenerate** on the **Fields** tab.

Fields Tab

The **Fields** tab defines the names, types, and positions of fields in the file. For more information, see:

Defining Fields In a Delimited Input File on page 198

Sort Tab

The **Sort** tab defines fields by which to sort the input records before they are sent into the dataflow. For more information, see **Sorting Records** on page 199.

Configuration Tab

To specify additional properties to run the required job, use this tab to define as many property-value pairs as required. You can add the required properties directly in the grid one at a time.

Alternatively, to import properties from a file, click **Import**. Go to the location of respective property file and select the file of XML format. The properties contained in the imported file are copied into the grid. The property file must be in XML format and must follow the syntax:

```
<configuration>
    <property>
        <name>key</name>
        <value>some_value</value>
        <description>A brief description of the
            purpose of the property key.</description>
    </property>
</configuration>
```

Note:

1. If the same property is defined here and in Management Console, the values defined here override the ones defined in Management Console.
2. If the same property exists both in the grid and also in the imported property file, then the value imported from the file overwrites the value existing in the grid for the same property.
3. You can import multiple property files one after the other, if required. The properties included in each imported file are added in the grid.
4. Ensure the property file is present on the Spectrum™ Technology Platform server itself.
5. The <description> tag is optional for each property key in a configuration property file.

6. Reference data needs to be placed local to data nodes to run the relevant jobs. This property is available only for jobs that use reference data, such as *Advanced Transformer*, *Validate Address Global*, and *Validate Address*. The property is: *pb.bdq.reference.data.location*.

Runtime Tab

Field Name	Description
File name	Displays the file name selected in the first tab.
Starting record	If you want to skip records at the beginning of the file when reading records into the dataflow, specify the first record you want to read. For example, if you want to skip the first 50 records, in a file, specify 51. The 51st record will be the first record read into the dataflow.
All records	Select this option if you want to read all records starting from the record specified in the **Starting record** field to the end of the file.
Max records	Select this option if you want to only read in a certain number of records starting from the record specified in the **Starting record** field. For example, if you want to read the first 100 records, select this option and enter 100.

Defining Fields In a Delimited Input File

The **Fields** tab defines the names, types, and positions of the fields in the file. After you define an input file on the **File Properties** tab you can define the fields.

If the input file does not contain a header record, or if you want to manually define the fields, follow these steps on the **Fields** tab:

1. To define the fields already present in the input file, click **Regenerate**. Then, click **Detect Type**. This will automatically set the data type for each field based on the first 50 records in the file.
2. To add additional fields in the output, click **Add**.
3. In the **Name** field, choose the field you want to add or type the name of the field.
4. In the **Type** field, you can leave the data type as `string` if you do not intend to perform any mathematical operations with the data. However, if you intend to perform such operations, select an appropriate data type. This will convert the string data from the file to a data type that will enable the proper manipulation of the data in the dataflow.

   The activity supports these data types:

bigdecimal   A numeric data type that supports 38 decimal points of precision. Use this data type for data that will be used in mathematical calculations requiring a high degree of precision, especially those involving financial data. The bigdecimal data type supports more precise calculations than the double data type.

double   A numeric data type that contains both negative and positive double precision numbers between 2^{-1074} and $(2-2^{-52}) \times 2^{1023}$. In E notation, the range of values is -1.79769313486232E+308 to 1.79769313486232E+308.

float   A numeric data type that contains both negative and positive single precision numbers between 2^{-149} and $(2-2^{23}) \times 2^{127}$. In E notation, the range of values -3.402823E+38 to 3.402823E+38.

integer   A numeric data type that contains both negative and positive whole numbers between -2^{31} (-2,147,483,648) and 2^{31}-1 (2,147,483,647).

long   A numeric data type that contains both negative and positive whole numbers between -2^{63} (-9,223,372,036,854,775,808) and 2^{63}-1 (9,223,372,036,854,775,807).

string   A sequence of characters.

5. In the **Position** field, enter the position of this field within the record.

   For example, in this input file, AddressLine1 is in position 1, City is in position 2, StateProvince is in position 3, and PostalCode is in position 4.

   ```
   "AddressLine1"|"City"|"StateProvince"|"PostalCode"
   "7200 13TH ST"|"MIAMI"|"FL"|"33144"
   "One Global View"|"Troy"|"NY"|12180
   ```

6. If you want to have any excess space characters removed from the beginning and end of a field's value string, select the **Trim** check box.

Sorting Records

The **Sort** tab allows you to define the criteria based on which the various fields must be sorted in the input records before they are sent into the dataflow.

1. On the **Sort** tab, click **Add**.
2. Click the drop-down arrow in the **Field Name** column and select the field you want to sort by. The fields available for selection depend on the fields defined in this input file.
3. In the **Order** column, select whether you want the field to be sorted in Ascending or Descending.
4. In the **Type** field, you can leave the data type as `string` if you do not intend to perform any mathematical or date time operations with the data. However, if you intend to perform these kinds of operations, select an appropriate data type. This will convert the `string` data from the file to a data type that will enable the proper manipulation of the data in the dataflow.
5. If you want to have any excess space characters removed from the beginning and end of a field's value string, select the **Trim** check box.

6. In the **Treat Null As** field, select whether you wish to treat `null` values of the particular field in any record as the `largest` or `smallest` value among all records.

7. Repeat until you have added all the input fields you wish to use for sorting. Change the order of the sort by highlighting the row for the field you wish to move and clicking **Up** or **Down**.

Variables

Inputs

In this grid, from among the fields received from a source activity, select the fields to be used in this activity.

Outputs

In this grid, select the fields to be included in the output to the destination activity in this process flow.

Submit Spark Job

The **Submit Spark Job** activity allows you to run any Spark job, either on a Hadoop cluster or on a Spark cluster. Using this activity, you can run either a Spark job of the Spectrum™ Big Data Quality SDK or any external Spark job.

Currently, you may submit a Spark job to either one of the two cluster types:

• YARN
• Spark

Deployment Modes

For a Spark job, you can use the *cluster* or *client* deployment modes. These deployment modes determine whether the Spark job driver class runs on the *cluster* or on the *client* Spectrum™ Technology Platform.

To simplify, you can run a Spark job in any one of the deployment modes:

1. YARN Cluster mode
2. YARN Client mode
3. Spark Client mode

> **Note:** The YARN or Spark *client* modes are recommended when the Spectrum server is installed and runs from within the cluster environment.

For a comprehensive list of supported job configurations on both Windows and Linux platforms, see the table **Supported job configurations** on page 208.

Field	Description
Job name	The name of the Spark job.

Field	Description
Hadoop server	The list of configured Hadoop servers. For information about mapping HDFS file servers through Management Console, see the *Administration Guide*.
Jar path	The path of the relevant JAR file for the Spark job to run. **Note:** The Jar path must point to a directory on the Spectrum server machine.
Job type	Select one of: **Spectrum** To run any one of the Spectrum Big Data Quality SDK jobs, select `Spectrum`. On selecting `Spectrum`, the field **Spectrum jobs** is displayed. **Generic** To specify additional properties for any external job, select `Generic`.

Field	Description
Spectrum jobs	Select the required job from the list of Spectrum Big Data Quality SDK jobs. The list includes these jobs:

<div></div>

- Address Validation
- Advanced Transformer
- Best of Breed
- Duplicate Syncronization
- Filter
- Groovy
- Intraflow Match
- Interflow Match
- Joiner
- Match Key Generator
- Open Name Parser
- Open Parser
- Table Lookup
- Transactional Match
- Validate Address
- Validate Address Global

On selecting the desired Spectrum job:

1. The fields **Job name**, **Class name**, and **Arguments** are auto-populated.

   All the auto-populated fields can be edited as required, except **Class name**.

   Important: For the selected Spectrum job, the auto-populated **Class name** must not be edited, else the job cannot run.

2. The **Properties** grid is auto-populated with the required configuration properties of the selected Spectrum job, with their default values.

   You can add or import more properties as well as modify the auto-populated properties, as required.

Field	Description
Class name	The fully qualified name of the driver class of the job.

Field	Description
Arguments	The space-separated list of arguments. These are passed to the driver class at runtime to run the job.

For example,

```
23Dec2016 /home/Hadoop/EYInc.txt
```

1. Those variables can be passed as arguments, which are defined to accept runtime values either in the source stage or this current stage of the process flow.

    For example, if in the output of the previous stage of the process flow the variable `SalesStartRange` is defined, you can include this variable in this argument list as `${SalesStartRange}` along with other required arguments, as illustrated:

    ```
    23Dec2016 /home/Hadoop/EYInc.txt ${SalesStartRange}
    ```

2. In case a particular argument contains a space, enclose it in double quotes. For example, `"/home/Hadoop/Sales Records"`.

Spectrum Big Data Quality SDK Jobs - Arguments:

To run the Spectrum Big Data Quality SDK *Spark* jobs, pass the various configuration files as a list of arguments. Each argument key accepts the path of a single configuration property file, where each file contains multiple configuration properties.

The syntax of the argument list for configuration properties is:

```
[-config <Path to configuration file>] [-debug] [-input <Path to
input configuration file>] [-conf <Path to Spark configuration file>]
[-output <Path of output directory>]
```

For example, for a Spark MatchKeyGenerator job:

```
-config /home/hadoop/spark/matchkey/matchKeyGeneratorConfig.xml -input
/home/hadoop/spark/matchkey/inputFileConfig.xml -output
/home/hadoop/spark/matchkey/outputFileConfig.xml
```

> **Note:** If the same configuration property key is specified both in the **Arguments** field and in the **Properties** grid but each points to different configuration files, the file indicated in the **Properties** grid for this property holds.

The sample configuration properties are shipped with the Data and Address Quality for Big Data SDK and are placed at the location `<Big Data Quality bundle>\samples\configuration`.

General Properties

Field	Description
Master	Select any of the options using which the Spark job is to run:
	YARN — To launch and manage the Spark job using YARN.
	Spark — To launch and manage the Spark job using a Spark application.
Spark URL	The URL to access the Spark cluster in the format `<hostname of Spark cluster>:<port of Spark cluster>`.
	This field becomes visible if you select `Spark` in the **Master** field .
Deploy-mode	Select any of the options:
	Client — To run the Spark job driver on the client Spectrum™ Technology Platform.
	Cluster — To run the Spark job driver on a cluster.

Field	Description
Properties	

Field	Description
	In the grid, under the **Property** column enter the names of the properties, and under the **Value** column enter the values of the corresponding properties.
	There are certain mandatory properties depending on the type of **Master** and **Deploy-mode**.

YARN Mandatory Properties	
`yarn.resourcemanager.hostname`	The IP address of the YARN ResourceManager.
`yarn.resourcemanager.address`	The address including the IP address and port of the YARN ResourceManager in the format `<hostname>:<port>`.

Client Deploy Mode Properties		
`Spark.driver.host`	The IP address of the machine on which the Spark driver is to run.	Required
`spark.client.mode.temp.location`	The path of the `temp` folder on the Spectrum server to be used for the Universal Addressing jobs: • Validate Address • Validate Address Global > **Note:** We strongly recommend using this property for the Universal Addressing jobs to ensure the specified `temp` folder is used for intermediate results.	Optional

Field	Description
	Thus: 1. For YARN cluster mode, the first two properties are mandatory. 2. For YARN client mode, all three properties are mandatory. 3. For SPARK client mode, the third property is mandatory. **Note:** You can define the above mandatory properties either while creating the connection in Management Console, or in this Spark activity. If the same properties are defined both in Management Console and also in the Spark Job activity, then the values assigned in the Spark activity are applicable. In addition to these mandatory properties, you can enter or import as many more properties as needed to run the job.
Import	To import properties from a file, click **Import**. Go to the location of respective property file and select the file of XML format. The properties contained in the imported file are copied into the **Properties** grid. **Note:** 1. If the same property is defined here and in Management Console, the values defined here override the ones defined in Management Console. 2. The property file must be in XML format and must follow the syntax: ```\n<configuration>\n <property>\n <name>key</name>\n <value>some_value</value>\n <description>A brief description of\nthe\n purpose of the property\nkey.</description>\n </property>\n</configuration>\n``` Create your own property files using the above XML format. 3. If the same property exists both in the grid and also in the imported property file, then the value imported from the file overwrites the value existing in the grid for the same property. 4. You can import multiple property files one after the other, if required. The properties included in each imported file are added in the grid. 5. Ensure the property file is present on the Spectrum™ Technology Platform server itself. 6. The `<description>` tag is optional for each property key in a configuration property file. 7. Reference data needs to be placed local to data nodes to run the relevant jobs. This property is available only for jobs that use reference data, such as *Advanced Transformer*, *Validate Address Global*, and *Validate Address*. The property is: *pb.bdq.reference.data.location*.

Dependencies

In this tab, add the list of the input files and Jar files required to run the job.

Once the job is run, the reference files and the reference Jar files added here are available in the distributed cache of the job.

Reference Files

> To add the various files required as input to run the job, click **Add**, go to the respective location on your local system or cluster, and select the particular file.

> To remove any file added in the list, select the particular file and click **Remove**.

Reference Jars

> To add the Jar files required to run the job, click **Add**, go to the respective location on your local system or cluster, and select the particular Jar file.

> To remove any file added in the list, select the particular file and click **Remove**.

> **Note:** The Jar path must point to a directory on the Spectrum server machine.

Supported job configurations

This table lists the supported Spark job configurations on Windows and Linux platforms:

Stages	Windows			Linux		
	Yarn+Client	Yam+Cluster	Spark+Client	Yarn+Client	Yarn+Cluster	Spark+Client
MatchKeyGenerator	Supported	Supported	Supported	Supported	Supported	Supported
Intraflow Match	Supported	Supported	Supported	Supported	Supported	Supported
Interflow Match	Supported	Supported	Supported	Supported	Supported	Supported
Transactional Match	Supported	Supported	Supported	Supported	Supported	Supported
Table Lookup	Supported	Supported	Supported	Supported	Supported	Supported
Advanced Transformer	Supported	Supported	Supported	Supported	Supported	Supported
Open Name Parser	Supported	Supported	Supported	Supported	Supported	Supported
Duplicate Synchronization	Supported	Supported	Supported	Supported	Supported	Supported

Stages	Windows			Linux		
	Yarn+Client	Yam+Cluster	Spark+Client	Yarn+Client	Yarn+Cluster	Spark+Client
Filter	Supported	Supported	Supported	Supported	Supported	Supported
Best of Breed	Supported	Supported	Supported	Supported	Supported	Supported
Open Parser	Supported	Supported	Supported	Supported	Supported	Supported
Address Validation (GAV)						
Groovy	Supported	Supported	Supported	Supported	Supported	Supported
Joiner	Supported	Supported	Supported	Supported	Supported	Supported
Validate Address Global (AD)	Supported	Supported	Supported	Supported	Supported	Supported
Validate Address (C1P)				Supported	Supported	Supported

Note:

- GAV and C1P stages are not supported on Windows platform where spectrum installation directory contains spaces.
- GAV and C1P stages behavior need to verify again on windows platform where spectrum installation directory does not contain any spaces.
- Reference data on HDFS strategy is not supported.
- Addressing job with *group by* feature is not supported in case of Yarn+Cluster mode.

Variables

Inputs

In this grid, from among the fields received from a source activity, select the fields to be used in this activity.

Outputs

In this grid, select the fields to be included in the output to the destination activity in this process flow.

Success

A Success activity indicates the end of a process flow. A process flow must have at least one Success activity.

6 - Creating Reusable Flow Components

In this section

Introduction to Subflows

A subflow is a dataflow that can be reused within other dataflows. Subflows are useful when you want to create a reusable process that can be easily incorporated into dataflows. For example, you might want to create a subflow that performs deduplication using certain settings in each stage so that you can use the same deduplication process in multiple dataflows. To do this you could create a subflow like this:

You could then use this subflow in a dataflow. For example, you could use the deduplication subflow within a dataflow that performs geocoding so that the data is deduplicated before the geocoding operation:

In this example, data would be read in from a database then passed to the deduplication subflow, where it would be processed through Match Key Generator, then Intraflow Match, then Best of Breed, and finally sent out of the subflow and on to the next stage in the parent dataflow, in this case Geocode US Address. Subflows are represented as a puzzle piece icon in the dataflow, as shown above.

Subflows that are saved and exposed are displayed in the **User Defined Stages** folder in Enterprise Designer.

Using a Subflow as a Source

You can use a subflow as the first stage in a dataflow to read data from a source and even perform some processing on the data before passing it to the parent dataflow. You can create a subflow that is as simple as a single source stage that is configured in a way that you want to reuse in multiple dataflows, or you could create a more complex subflow that reads data and then processes it in some way before passing it to the parent dataflow.

1. In Enterprise Designer, click **File** > **New** > **Dataflow** > **Subflow**.
2. Drag the appropriate data source from the palette onto the canvas and configure it.

For example, if you want the subflow to read data from a comma-separated file, you would drag a Read from File stage onto the canvas.

3. If you want the subflow to process the data in some way before sending it to the parent dataflow, add additional stages as needed to perform the preprocessing you want.

4. At the end of the dataflow, add an Output stage and configure it.

This allows the data from the subflow to be sent to the parent dataflow.

For example, if you created a subflow that reads data from a file then uses a Transformer stage to trim white space and standardize the casing of a field, you would have a subflow that looks like this:

5. Double-click the Output stage and select the fields you want to pass into the parent dataflow.

6. Select **File** > **Save** and save the subflow.

7. Select **File** > **Expose** to make the subflow available to include in dataflows.

8. In the dataflow where you want to include the subflow, drag the subflow from the palette onto the canvas.

9. Connect the subflow to the dataflow stage you want.

> **Note:** Since the subflow contains a source stage rather than an Input stage, the subflow icon only has an output port. It can only be used as a source in the dataflow.

The parent dataflow now uses the subflow you created as input. For example, if you created a subflow named "Read from File and Transform" and you add the subflow and connect it to a Broadcaster stage, your dataflow would look like this:

Using a Subflow in the Middle of a Dataflow

You can use a subflow in the middle of a dataflow to perform processing that you want to make reusable in other dataflows. In effect, the subflow becomes a custom stage in your dataflow.

1. In Enterprise Designer, click **File** > **New** > **Dataflow** > **Subflow**.

2. Drag an Input stage from the palette to the canvas.

This allows data from the parent dataflow to be sent into the subflow.

3. Double-click the Input stage and add the fields that the subflow will receive from the dataflow in which it is used.

4. After configuring the Input stage, add additional stages as needed to perform the processing that you want.

5. At the end of the dataflow, add an Output stage.

   This allows the data from the subflow to be sent to the parent dataflow.

   For example, you might want to create a subflow that performs deduplication using certain settings in each stage so that you can use the same deduplication process in multiple dataflows. To do this you could create a subflow like this:

6. Select **File** > **Save** and save the subflow.

7. Select **File** > **Expose** to make the subflow available to include in dataflows.

8. In the dataflow where you want to include the subflow, drag the subflow from the palette onto the canvas.

9. Connect the subflow to the dataflow stage you want.

   For example, you could use the deduplication subflow within a dataflow that performs geocoding so that the data is deduplicated before the geocoding operation:

Using a Subflow as a Sink

You can use a subflow as the last stage in a dataflow to write data to a file or database and even perform some processing on the data before writing the data to the output destination. You can create a subflow as simple as a single sink stage that is configured in a way that you want to reuse in multiple dataflows, or you could create a more complex subflow that processes data in some way before writing it to the output destination.

1. In Enterprise Designer, click **File** > **New** > **Dataflow** > **Subflow**.

2. Drag an Input stage from the palette to the canvas.

3. Double-click the Input stage and add the fields that the subflow will receive from the dataflow in which it is used.

4. After configuring the Input stage, add additional stages as needed to perform the post-processing that you want.

5. At the end of the subflow, add the appropriate sink.

   For example, if you created a subflow that uses a Transformer stage to trim white space and standardize the casing of a field then writes it to a database, you would have a subflow that looks like this:

6. Select **File** > **Save** and save the subflow.

7. Select **File** > **Expose** to make the subflow available to include in dataflows.

8. In the dataflow where you want to include the subflow, drag the subflow from the palette onto the canvas and connect it to the last stage in the dataflow.

   > **Note:** Since the subflow contains a sink stage rather than an Output stage, the subflow icon only has an input port. It can only be used as a sink in the dataflow.

   The parent dataflow now uses the subflow you created as a sink. For example, if you created a subflow named "Transform and Write to DB" and you add the subflow and connect it to a Geocode US Address stage, your dataflow would look like this:

Modifying a Subflow

1. Open the subflow in Enterprise Designer.

2. Before modifying the subflow, you may want to consider how the change will impact the dataflows using the subflow. To see which dataflows are using the subflow, select **Tools** > **Used By**.

3. Modify the subflow as needed.

   Note the following:

   • When you delete an Input or Output stage or add an additional Input or Output stage, Enterprise Designer displays a warning message reminding you that other dataflows are using the subflow and giving you the option of seeing which dataflows use the subflow. If you continue saving the reusable stage, Enterprise Designer will unexpose all dataflows used by the subflow.

- If you change a subflow in another way, such as by changing a file name or the stage configurations, Enterprise Designer will display a warning message reminding you that other dataflows are using the subflow and give you the option of seeing which dataflows use the subflow. You can continue without unexposing those dataflows.

4. When you are done making your changes, select **File** > **Save**.
5. Select **View** > **Refresh** in order for the changes to be reflected in the parent dataflow.

> **Note:** If you have more than one version of the subflow, remember that the version that is used in the parent dataflow is the exposed version. When you make a change to a subflow, be sure to expose the most recent version in order for your changes to take effect in the dataflows that use the subflow.

Deleting a Subflow

If you try to delete an exposed subflow, Enterprise Designer displays a warning message reminding you that other dataflows are using the subflow you are about to delete. If you continue to delete the subflow, Enterprise Designer unexposes all connected dataflows.

Exposing and Unexposing a Subflow

In order for a subflow to be available for use within a dataflow the subflow must be exposed. To expose a subflow, open the subflow in Enterprise Designer and go to **File** > **Expose/Unexpose and Save**. This will make the subflow available for use in other dataflows.

> **Note:** If you have more than one version of the subflow, remember that the version that is used in the parent dataflow is the exposed version. When you make a change to a subflow, be sure to expose the most recent version in order for your changes to take effect in the dataflows that use the subflow.

To unexpose a subflow, open the subflow in Enterprise Designer and select **File** > **Expose/Unexpose and Save**. When you unexpose a subflow, Enterprise Designer displays a warning message reminding you that other dataflows are using the subflow you are about to alter. If you continue to unexpose the subflow, Enterprise Designer unexposes all dataflows that use the subflow.

Converting a Stage to a Subflow

1. Create a new job, service, or subflow.
2. Add the stage you would like to include in the job, service, or subflow.
3. If you wish to configure the stage at this point, right-click the stage and select **Options**. Then configure the stage options as desired and click **OK**.
4. Right-click the stage you want to convert and select **Convert Stage to Subflow**. The **Save As** dialog box appears.
5. Enter the name you want to give the subflow and click **OK**, then save the service. The name must be unique to the system. Three things happen:

   • The system creates a new subflow that includes the following:

     • the stage you selected
     • a dataflow input for each input port on the stage
     • a dataflow output for each output port on the stage
     • connections between the stage and its inputs and outputs

   • The system replaces your selected stage with the new subflow.

   • The system exposes the new subflow. You will see it in the Server Explorer and in the User Defined Stages section of the toolbox.

After you have created a subflow and used it in other dataflows, you can see what other dataflows are using the subflow. Open the subflow and go to **Tools** > **Used By**. (Alternately, you can right-click the subflow in Server Explorer and select **Used By**.) This will show a list of dataflows that use the current subflow, allowing you to see which dataflows would be affected if you changed the current subflow.

7 - Sample Flows

In this section

Introduction

Sample Template is a set of pre-configured templates using which you can demonstrate Spectrum migration capabilities. It eliminitates the need to create data flows from scratch. You need to create a couple of connections and import these flows into your system to demonstrate the migration capabilities. This section describes the end-to-end process of data migration using sample flows.

Locating the Sample Template

The sample template is shipped with the Spectrum Technology Platform installer zip. Depending on your installation directory, you can find the sample template at the following path:

```
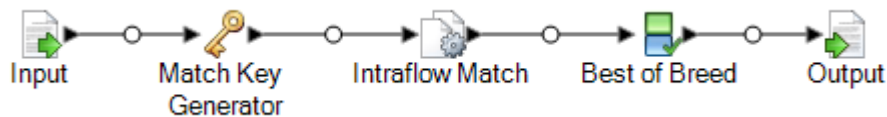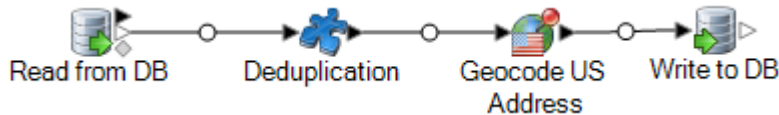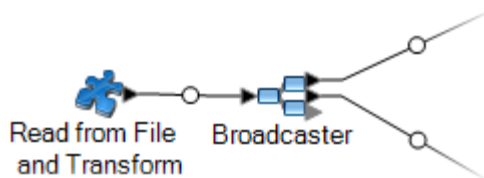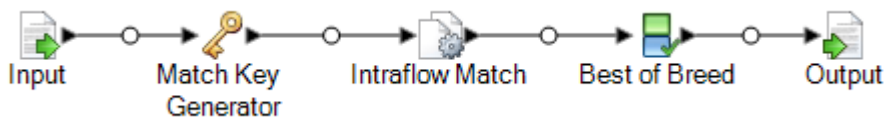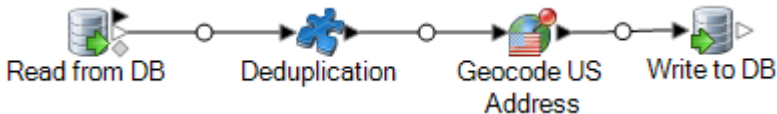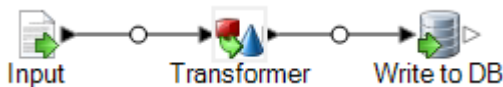Program Files\Pitney
Bowes\Spectrum\server\modules\metadata-insights\connectors\samples
```

The folder contains all the required files to configure the data flow.

Creating Connections

You need to create two connections to import and deploy required files to configure the data flow. Follow these steps to create connections:

1. Install the latest Spectrum platform Server
2. Open the Spectrum Server home page
3. Click **Platform Client Tools** and then click **Web**
4. Open the **Management Console** view
5. Click **Resources** menu and then click **Data Sources** and create two connections.

Importing and Deploying Files to the Server

With connection in place, now you need a command line utility to import and deploy the files to the server.

1. Open the Spectrum platform home page, click **Platform Client Tools** followed by **Command Line** and download the following files:

   a. Job Executor, which is an individual .jar file
   b. Administration Utility, which is a zipped file (spectrum-cli)

2. Unzip the spectrum-cli file
3. Copy all the files from the Sample Template folder to the spectrum-cli folder. This saves you from entering full path of every file with each command
4. Run cli.cmd utility located in the spectrum-cli-12.1 folder. The command line interface of the Spectrum Platform is launched

5. Connect to the Spectrum Server using following commands: connect <server name>:<port> --u <username> --p <password>. Successful connection is confirmed by message: Connected to server<server name> :< port>

6. Type and execute the following commands in the given order to import and deploy the sample data flows:

   a. Import modelstore using command: `spectrum> modelstore bulkImport --importDependency true` This command imports the modelstore with all the dependencies.

   b. Deploy modelstore using command:`modelstore deploy --n <name>`

   c. Import dbconnection using command: `dbconnection import --f <name.json>`

   d. Import data flows using command: `dataflow import --f <name.df>`

   Note:

   Remove any extra spaces around commands of filenames

   Replace the placeholder <name> with the exact name of the file

Viewing Imported Files

After successful import, you can use these data flows to demonstrate the Spectrum capabilities. Files are stored at following locations:

1. **Modelstore:** In the Physical Model and Model Store tabs in Metadata Insights
2. **Dataflow:** In the Server Explorer of Enterprise Designer application
3. **Json:** In the Model Store in Management Console

Imported flows can be used to migrate data from one system to other using Enterprise Designer. Detailed steps for each of the flows is mentioned in their respetive following secitons.

   Note: These sample templates are part of the Spectrum installer file and do not work standalone. There is no further support available for these samples.

Integration between SugarCRM OnPremises and Microsoft Dynamics 365 Online

This section illustrates the migration of Account and Contacts from SugarCRM OnPremises to Microsoft Dynamics 365 Online systems. If there is an existing association of Contacts and Account in the SugarCRM, the migration takes care to maintain the same in Microsoft Dynamics 365 Online system as well.

The Sample Template contains following files:

Table 10: Files in Sample Template folder

File Type	File Name
Data Flow	• SugarCRMAccount_Sync_MSDAccount.df • SugarCRMContact_Sync_MDSContact.df
DB Connection	• MSDynamics_MS.json • SugarCRM_MS.json
ModelStore	• mi_modelStore_MSDynamics_MS.smims • mi_modelStore_SugarCRM_MS.smims

Do the following to migrate data from SugarCRM OnPremise to MS Dynamics 365 Online

1. Create following connections:

   a. **SugarCRM_TestConnection** with Type SugarCRM. Click Test and a success message is displayed: *The connection SugarCRM_OnPremises successfully connected to the data source*

   b. **MSDynamics_TestConnection** with Type Microsoft Dynamics 365. Click Test and a success message is displayed: Success: *The connection Microsoft Dynamics 365 Online successfully connected to the data source*

   For more details on how to create connections, refer to **Connecting to SugarCRM** and **Connecting to Microsoft Dynamics 365 Online**.

2. Import and deploy files using Spectrum Command line utility. Use following commands in given order:

   • Import modelstore using command:

   ```
   modelstore bulkimport --importDependency true
   ```

   • Deploy modelstore using command:

   ```
   modelstore deploy --modelStoreName MSDynamics_MS
   modelstore deploy --modelStoreName SugarCRM_MS
   ```

   • Import dbconnection using command:

   ```
   dbconnection import --f MSDynamics_MS.json
   dbconnection import --f SugarCRM_MS.json
   ```

   • Import data flows using command:

   ```
   dataflow import --f SugarCRMAccount_Sync_MSDAccount.df
   dataflow import --f SugarCRMContact_Sync_MDSContact.df
   ```

3. Launch the Enterprise Designer application. You can download the executable setup of this application from **Desktop** section of **Platform Client Tool** on the Spectrum Platform home page

4. Login using your Spectrum credentials

5. Click **View** from the menu and then click **Server Explorer**

6. Double-click the **SugarCRMAccount_Sync_MSDAccount** dataflow job first

7. Double-click Read from **DB_SugarCRM** stage

   a. Change the value of field date_entered as required and click **OK**

   ```
   Select "SugarCRM_PM"."Accounts"."email1",
   "SugarCRM_PM"."Accounts"."name",
   "SugarCRM_PM"."Accounts"."phone_office",
   "SugarCRM_PM"."Accounts"."date_entered" From
    "SugarCRM_PM"."Accounts" Where
   "SugarCRM_PM"."Accounts"."date_entered" Like '2017-08-28%'
   ```

   b. Click **OK** to continue

8. Run the flow by clicking **Run** button on the Tool Bar in Enterprise Designer

9. Visit the MS Dynamics home page and click **Accounts** under **Customers** tab on the **Sales** page. Notice that the **Accounts** have been migrated

10. Double-click the **SugarCRMContact_Sync_MDSContacts** job

11. Double-click **Read from_MSDynamics_DB** stage

    a. Change the value of the field createdon as required and click **OK**

    ```
    Select "MSDynamics_PM"."account"."name",
    "MSDynamics_PM"."account"."telephone1",
     "MSDynamics_PM"."account"."emailaddress1",
    "MSDynamics_PM"."account"."createdon",
     "MSDynamics_PM"."account"."accountid" From
    "MSDynamics_PM"."account"
     Where "MSDynamics_PM"."account"."createdon" Like '2017-09-11%'
    ```

    b. Click **OK** to continue

12. Run the flow by clicking **Run** button on the Tool Bar in Enterprise Designer

13. Visit the MS Dynamics home page and click **Contacts** under **Customers** tab on the **Sales** page. Notice that the **Contacts** are listed on the page

The successful listing of accounts and contacts in the MS Dynamics page complete the migration process

Integration between Salesforce and Oracle Eloqua

This section illustrates the migration of Account and Contacts from Salesforce to Oracle Eloqua systems. If there is an existing association of Contacts and Account in the Salesforce, the migration takes care to maintain the same in Oracle Eloqua system as well.

The sample template contains these files:

Table 11: Files in Sample Template folder

File Type	File Name
Data Flow	• *SalesforceAccount_Sync_OracleEloquaAccount.df* • *SalesforceContact_Sync_OracleEloquaContact.df*
DB Connection	• *Eloqua_MS.json* • *Salesforce_MS.json*
ModelStore	• *mi_modelStore_Eloqua_MS.smims* • *mi_modelStore_Salesforce_MS.smims*

To migrate data from Salesforce to Oracle Eloqua, perform these steps:

1. Create these connections:

   a. **Salesforce_TestConnection** with Type Salesforce. Click Test and a success message is displayed: *The connection Salesforce successfully connected to the data source*

   b. **Eloqua_TestConnection** with Type Oracle Eloqua. Click Test and a success message is displayed: Success: *The connection Eloqua successfully connected to the data source*

   For more details on how to create connections, refer to **Connecting to Salesforce** and **Connecting to Oracle Eloqua**.

2. Import and deploy files using Spectrum Command line utility. Use following commands in given order:

   • Import modelstore using command:

   ```
   modelstore bulkimport --importDependency true
   ```

   • Deploy modelstore using command:

```
modelstore deploy --modelStoreName Eloqua_MS
modelstore deploy --modelStoreName Salesforce_MS
```

• Import dbconnection using command:

```
dbconnection import --f Eloqua_MS.json
dbconnection import --f Salesforce_MS.json
```

• Import data flows using command:#

```
dataflow import --f SalesforceAccount_Sync_OracleEloquaAccount.df
dataflow import --f SalesforceContact_Sync_OracleEloquaContact.df
```

3. Launch the Enterprise Designer application. You can download the executable setup of this application from **Desktop** section of **Platform Client Tool** on the Spectrum Platform home page

4. Login using your Spectrum credentials

5. Click **View** from the menu, click **Server Explorer**.

6. Double-click the **SalesforceAccount_Sync_OracleEloquaAccount** dataflow job first.

7. Double-click **Read from DB** stage.

    a. Change the value of field CreatedDate as required and click **OK**.

```
Select "SalesforcePM"."Account".* From "SalesforcePM"."Account"
where CreatedDate  > '2017-01-01'
```

    b. Click **OK** to continue.

8. Run the flow by clicking **Run** button on the Tool Bar in Enterprise Designer.

9. Visit the Eloqua home page and click **Account**. Notice the accounts have mnigrated to Eloqua.

10. Navigate back to Enterprise Designer.

11. Double-click the **SalesforceContact_Sync_OracleEloquaContact** job.

12. Double-click **Read from DB** stage.

    a. Change the value of the field CreatedDate as required and click **OK**.

```
Select "SalesforcePM"."Contact".* From "SalesforcePM"."Contact"
where CreatedDate  > '2017-01-01'
```

    b. Click **OK** to continue.

13. Run the flow by clicking **Run** button on the Tool Bar in Enterprise Designer.

14. Visit the Eloqua home page and click **Contacts**. Notice the contacts have mnigrated to Eloqua.

The successful listing of accounts and contacts in the Oracle Eloqua complete the migration process.

8 - About Spectrum™ Technology Platform

In this section

What Is Spectrum™ Technology Platform?

Spectrum™ Technology Platform is a system that improves the completeness, validity, consistency, timeliness, and accuracy of your data through data standardization, verification and enhancement. Ensuring that your data is accurate, complete, and up to date enables your firm to better understand and connect with your customers.

Spectrum™ Technology Platform aids in the design and implementation of business rules for data quality by performing the following functions.

Parsing, Name Standardization, and Name Validation

To perform the most accurate standardization you may need to break up strings of data into multiple fields. Spectrum™ Technology Platform provides advanced parsing features that enable you to parse personal names, company names, and many other terms and abbreviations. In addition, you can create your own list of custom terms to use as the basis of scan/extract operations. The Universal Name Module provides this functionality.

Deduplication and Consolidation

Identifying unique entities enables you to consolidate records, eliminate duplicates and develop "best-of-breed" records. A "best-of-breed" record is a composite record that is built using data from other records. The Advanced Matching Module and Data Normalization Module provide this functionality.

Address Validation

Address validation applies rules from the appropriate postal authority to put an address into a standard form and even validate that the address is a deliverable address. Address validation can help you qualify for postal discounts and can improve the deliverability of your mail. The Universal Addressing Module and the Address Now Module provide this functionality.

Geocoding

Geocoding is the process of taking an address and determining its geographic coordinates (latitude and longitude). Geocoding can be used for map generation, but that is only one application. The underlying location data can help drive business decisions. Reversing the process, you can enter a geocode (a point represented by a latitude and longitude coordinate) and receive address information about the geocode. The Enterprise Geocoding Module provides this functionality.

Location Intelligence

Location intelligence creates new information about your data by assessing, evaluating, analyzing and modeling geographic relationships. Using location intelligence processing you can verify locations

and transform information into valuable business intelligence. The Location Intelligence Module provides this functionality.

Master Data Management

Master data management enables you to create relationship-centric master data views of your critical data assets. The Data Hub Module helps you identify influencers and non-obvious relationships, detect fraud, and improve the quality, integration, and accessibility of your information.

Tax Jurisdiction Assignment

Tax jurisdiction assignment takes an address and determines the tax jurisdictions that apply to the address's location. Assigning the most accurate tax jurisdictions can reduce financial risk and regulatory liability.

Spectrum™ Technology Platform software from Pitney Bowes integrates up-to-date jurisdictional boundaries with the exact street addresses of your customer records, enabling you to append the correct state, county, township, municipal, and special tax district information to your records. Some example uses of tax jurisdiction assignment are:

• Sales and use tax
• Personal property tax
• Insurance premium tax

The Enterprise Tax Module provides this functionality.

Enterprise Data Management Architecture

With Spectrum™ Technology Platform, you can build a comprehensive enterprise data management process, or you can use it as a more targeted solution. The following diagram illustrates a complete solution that takes data from its source, through data enrichment and data quality processes, feeding a master data management hub which makes a single view of the data available to multiple business applications.

Data Discovery

Data discovery is the process of scanning your data resources to get a complete inventory of your data landscape. Spectrum™ Technology Platform can scan structured data, unstructured data, and semi-structured data using a wide array of data profiling techniques. The results of the scan are used to automatically generate a library of documentation describing your company's data assets and to create a metadata repository. This documentation and accompanying metadata repository provide the insight you need before beginning data integration, data quality, data governance, or master data management projects.

For more information on the Spectrum™ Technology Platform Data Discovery Module, contact your account executive.

Data Integration

Once you have an inventory of your data landscape, you need to consider how you will access the data you need to manage. Spectrum™ Technology Platform can connect to data in multiple sources either directly or through integration with your existing data access technologies. It supports batch and real time data integration capabilities for a variety of business needs including data warehousing, data quality, systems integration, and migration. Spectrum™ Technology Platform can access data in RDBMS databases, data warehouses, XML files, flat files, and more. Spectrum™ Technology Platform supports SQL queries with complex joins and aggregations and provides a visual query development tool. In addition, Spectrum™ Technology Platform can access data over REST and SOAP web services.

Spectrum™ Technology Platform can trigger batch processing based on the appearance of one or more source files in a specified folder. This "hot folder" trigger is useful for monitoring FTP uploads and processing them as they occur.

Some of these data integration capabilities require a license for the Enterprise Data Integration Module. For more information, contact your account executive.

Finally, Spectrum™ Technology Platform can integrate with packaged applications such as SAP.

Data Quality/Governance

Data quality and data governance processes check your data for duplicate records, inconsistent information, and inaccurate information.

Duplicate matching identifies potential duplicate records or relationships between records, whether the data is name and address in nature or any other type of customer information. Spectrum™ Technology Platform allows you to specify a consistent set of business match rules using boolean matching methods, scoring methods, thresholds, algorithms and weights to determine if a group of records contains duplicates. Spectrum™ Technology Platform supports extensive customization so you can tailor the rules to the unique needs of your business.

Once duplicate records have been identified, you may wish to consolidate records. Spectrum™ Technology Platform allows you to specify how to link or merge duplicate records so you can create the most accurate and complete record from any collection of customer information. For example, a single best-of-breed record can be built from all of the records in a household. The Advanced Matching Module is used to identify duplicates and eliminate them.

Data quality processes also standardize your data. Standardization is a critical process because standardized data elements are necessary to achieve the highest possible results for matching and identifying relationships between records. While several modules perform standardization of one type or another, the Spectrum™ Technology Platform Data Normalization module provides the most comprehensive set of standardization features. In addition, the Universal Name module provides specific data quality features for handling personal name and business name data.

Standardized data is not necessarily accurate data. Spectrum™ Technology Platform can compare your data to known, up-to-date reference data for correctness. The sources used for this process may include regulatory bodies such as the U.S. Postal Service, third-party data providers such as Experian or D&B, or your company's internal reference sources, such as accounting data. Spectrum™

Technology Platform is particularly strong in address data validation. It can validate or standardize addresses in 250 countries and territories around the world. There are two modules that perform address validation: the Address Now Module and the Universal Addressing Module.

To determine which one is right for you, discuss your needs with your account executive.

While Spectrum™ Technology Platform can automatically handle a wide range of data quality issues, there are some situations where a manual review by a data steward is appropriate. To support this, the Business Steward Module provides a way to specify the rules that will trigger a manual review, and it provides a web-based tool for reviewing exception records. It includes integrated access to third-party tools such as Bing maps and Experian data to aid data stewards in the review and resolution process.

Data Enrichment

Data enrichment processes augment your data with additional information. Enrichment can be based on spatial data, marketing data, or data from other sources that you wish to use to add additional detail to your data. For example, if you have a database of customer addresses, you could geocode the address to determine the latitude/longitude coordinates of the address and store those coordinates as part of the record. Your customer data could then be used to perform a variety of spatial calculations, such as finding the bank branch nearest the customer. Spectrum™ Technology Platform allows you to enrich your data with a variety of information, including geocoding (with the Enterprise Geocoding Module), tax jurisdiction assignment (with the Enterprise Tax Module), geospatial calculations (with the Location Intelligence Module), and driving and walking directions between points (with the Enterprise Routing Module).

Master Data Management Hub

The Master Data Management (MDM) hub allows for rapid modeling of entities and their complex relationships across roles, processes and interactions. It provides built-in social network analysis capabilities to help you understand influencers, predict churn, detect non-obvious relationships and fraudulent patterns, and provide recommendations.

Spectrum™ Technology Platform supports two approaches to the MDM hub. In the master hub approach, the data is maintained in a single MDM database and applications access the data from the MDM database. In the registry approach, the data is maintained in each business application and the MDM hub registry contains keys which are used to find related records. For example, a customer's record may exist in an order entry database and a customer support database. The MDM registry would contain a single key which could be used to access the customer data in both places.

The Data Hub Module provides MDM capabilities.

Spectrum™ Technology Platform Architecture

Spectrum™ Technology Platform from Pitney Bowes consists of a server that runs a number of modules. These modules provide different functions, such as address validation, geocoding, and advanced parsing, among others. The following diagram illustrates the Spectrum™ Technology Platform architecture.

Server

The foundation of the Spectrum™ Technology Platform is the server. The server handles data processing, synchronizes repository data, and manages communication. It provides job management and security features.

Modules

Modules are sets of features that perform a specific function. For example, the Universal Addressing Module standardizes addresses to conform to postal standards. The Enterprise Tax Module determines the tax jurisdictions that apply to a given address. Modules are grouped together to solve common business problems and licensed together as bundles.

Components

Modules are comprised of components which perform a specific function in a flow or as a service. For example, the Enterprise Geocoding module's Geocode US Address component takes an address and returns the latitude and longitude coordinates for that address; the Universal Addressing module's Get City State Province takes a postal code and returns the city and state or province where that postal code is located.

The components that you have available on your system depend on which Spectrum™ Technology Platform bundle you have licensed.

Databases

Some modules depend on databases containing reference data. For example, the Universal Addressing module needs to have access to U.S. Postal Service data in order to verify and standardize addresses in the U.S. Databases are installed separately and some are updated on a regular basis to provide you with the latest data.

Modules have both required and optional databases. Optional databases provide data needed for certain features that can enhance your Spectrum™ Technology Platform process.

Management Console

Management Console is a tool for administering Spectrum™ Technology Platform. You can use Management Console to:

- Define the connections between Spectrum™ Technology Platform and your data
- Specify the default settings for services and flows
- Manage user accounts, including permissions and passwords
- View logs
- View licenses including license expiration information

Enterprise Designer

Enterprise Designer is a tool for creating Spectrum™ Technology Platform jobs, services, subflows, and process flows. It utilizes an easy drag-and-drop interface to allow you to graphically create complex dataflows.

Metadata Insights

Metadata Insights gives you the control you need to deliver accurate and timely data-driven insights to your business. Use Metadata Insights to develop data models, view the flow of data from source to business application, and assess the quality of your data through profiling. With this insight, you can identify the data resources to use to answer particular business questions, adapt and optimize processes to improve the usefulness and consistency of data across your business, and troubleshoot data issues.

Web Services and API

You can integrate Spectrum™ Technology Platform capabilities into your applications using web services and programming APIs. These interfaces provide simple integration, streamline record processing, and support backward compatibility of future versions.

The Spectrum™ Technology Platform API is available for these languages:

• C
• C++
• COM
• Java
• .NET

Web services are available via SOAP and REST.

Modules and Components

Table 12: Modules and Components

Module	Description	Components
Advanced Matching Module	Matches records within and/or between input files.	Best Of Breed Candidate Finder Duplicate Synchronization Filter Interflow Match Intraflow Match Match Key Generator Transactional Match
Business Steward Module	Identifies exception records and provides a browser-based tool for manually reviewing exception records.	Exception Monitor Read Exceptions Write Exceptions
Country Identifier	Takes a country name or a combination of postal code and state/province and returns the two-character ISO country code, the three-character Universal Postal Union (UPU) code, and the English country name.	Country Identifier
Metadata Insights	Gives you the control you need to deliver accurate and timely data-driven insights to your business. Develops data models, gives you a view the flow of data from source to business application, and assesses the quality of your data through profiling. It helps you identify the data resources you should use to answer particular business questions and to optimize processes to improve the usefulness and consistency of data across your business.	Models (Logical and Physical) Model Store Profile Lineage and Impact Analysis

Module	Description	Components
Data Hub Module	Links and analyzes data, identifying relationships and trends.	Write to Hub Read From Hub Query Hub Graph Visualization
Data Integration Module	Provides capabilities useful in data warehousing, data quality, systems integration, and migration.	Field Selector Generate Time Dimension Query Cache Write to Cache
Data Normalization Module	Removes inconsistencies in data.	Advanced Transformer Open Parser Table Lookup Transliterator
Enterprise Data Integration	Connects to data in multiple sources for a variety of business needs including data warehousing, data quality, systems integration, and migration.	Call Stored Procedure Field Selector Generate Time Dimension Query Cache Write to Cache
Enterprise Geocoding Module	Determines the geographic coordinates for an address. Also determines the address of a given latitude and longitude.	Geocode Address AUS Geocode Address GBR - deprecated. Use Global Geocoding Module geocoding stage. Geocode Address Global Geocode Address World Geocode US Address GNAF PID Location Search Reverse APN Lookup Reverse Geocode Address Global Reverse Geocode US Location

Module	Description	Components
Enterprise Routing Module	Obtains driving or walking directions, calculates drive time and drive distance, and identifies locations within a certain time or distance from a starting point.	Get Route Data Get Travel Boundary Get Travel Cost Matrix Get Travel Directions Persistent Update
Enterprise Tax Module	Determines the tax jurisdictions that apply to a given location.	Assign GeoTAX Info Calculate Distance
FCC Screening Module	Helps banks and financial institutions to effectively detect financial crimes, reduce false positives, and maintain robust detection capability as required by the regulators.	Party Groups Lists Screen Alerts
GeoConfidence Module	Determines the probability that an address or street intersection is within a given area.	Geo Confidence Surface CreatePointsConvexHull
Global Addressing Module	Provides enhanced address standardization and validation. Also, automatically suggests addresses as you type and immediately returns candidates based on your input. Splits postal address strings into individual address elements using machine learning techniques.	Global Address Parser Global Address Validation Global Type Ahead
Global Geocoding Module	Determines the geographic coordinates for an address. Also determines the address of a given latitude and longitude. Interactive geocoding is a type-ahead feature in GGM. Key Lookup uses a key to geocode addresses.	Global Geocode Global Reverse Geocode Global Interactive Geocoding Global Key Lookup

Module	Description	Components
Global Sentry	Attempts to match transactions against government-provided watch lists that contain data from different countries.	Global Sentry Global Sentry Address Check Global Sentry ID Number Check Global Sentry Name Check Global Sentry Other Data Check
Location Intelligence Module	Performs point in polygon and radial analysis against a variety of geospatial databases.	Closest Site Find Nearest Point In Polygon Query Spatial Data Read Spatial Data Spatial Calculator Spatial Union Write Spatial Data
SAP Module	Enables Spectrum™ Technology Platform to interface with SAP Customer Relationship Management Module applications.	SAP Generate Match Key SAP Generate Match Score SAP Generate Search Key SAP Generate Search Key Constant SAP Generate Search Key Metaphone SAP Generate Search Key Substring SAP Validate Address With Candidates
Universal Addressing Module	Standardizes and validates addresses according to the postal authority's standards.	Get Candidate Addresses Get City State Province Get Postal Codes Validate Address Validate Address AUS Validate Address Global

Module	Description	Components
Universal Name Module	Parses personal names, company names, addresses, and many other terms and abbreviations.	Name Parser (Deprecated) Name Variant Finder Open Name Parser

Notices

USPS® Notices

Pitney Bowes Inc. holds a non-exclusive license to publish and sell ZIP + 4® databases on optical and magnetic media. The following trademarks are owned by the United States Postal Service: CASS, CASS Certified, DPV, eLOT, FASTforward, First-Class Mail, Intelligent Mail, LACS^{Link}, NCOA^{Link}, PAVE, PLANET Code, Postal Service, POSTNET, Post Office, RDI, Suite^{Link} , United States Postal Service, Standard Mail, United States Post Office, USPS, ZIP Code, and ZIP + 4. This list is not exhaustive of the trademarks belonging to the Postal Service.

Pitney Bowes Inc. is a non-exclusive licensee of USPS® for NCOA^{Link}® processing.

Prices for Pitney Bowes Software's products, options, and services are not established, controlled, or approved by USPS® or United States Government. When utilizing RDI™ data to determine parcel-shipping costs, the business decision on which parcel delivery company to use is not made by the USPS® or United States Government.

Data Provider and Related Notices

Data Products contained on this media and used within Pitney Bowes Software applications are protected by various trademarks and by one or more of the following copyrights:

© Copyright United States Postal Service. All rights reserved.

© 2014 TomTom. All rights reserved. TomTom and the TomTom logo are registered trademarks of TomTom N.V.

© 2016 HERE

Fuente: INEGI (Instituto Nacional de Estadística y Geografía)

Based upon electronic data © National Land Survey Sweden.

© Copyright United States Census Bureau

© Copyright Nova Marketing Group, Inc.

Portions of this program are © Copyright 1993-2007 by Nova Marketing Group Inc. All Rights Reserved

© Copyright Second Decimal, LLC

© Copyright Canada Post Corporation

This CD-ROM contains data from a compilation in which Canada Post Corporation is the copyright owner.

© 2007 Claritas, Inc.

The Geocode Address World data set contains data licensed from the GeoNames Project (**www.geonames.org**) provided under the Creative Commons Attribution License ("Attribution License") located at **http://creativecommons.org/licenses/by/3.0/legalcode**. Your use of the

GeoNames data (described in the Spectrum™ Technology Platform User Manual) is governed by the terms of the Attribution License, and any conflict between your agreement with Pitney Bowes Software, Inc. and the Attribution License will be resolved in favor of the Attribution License solely as it relates to your use of the GeoNames data.

pitney bowes

3001 Summer Street
Stamford CT 06926-0700
USA

www.pitneybowes.com